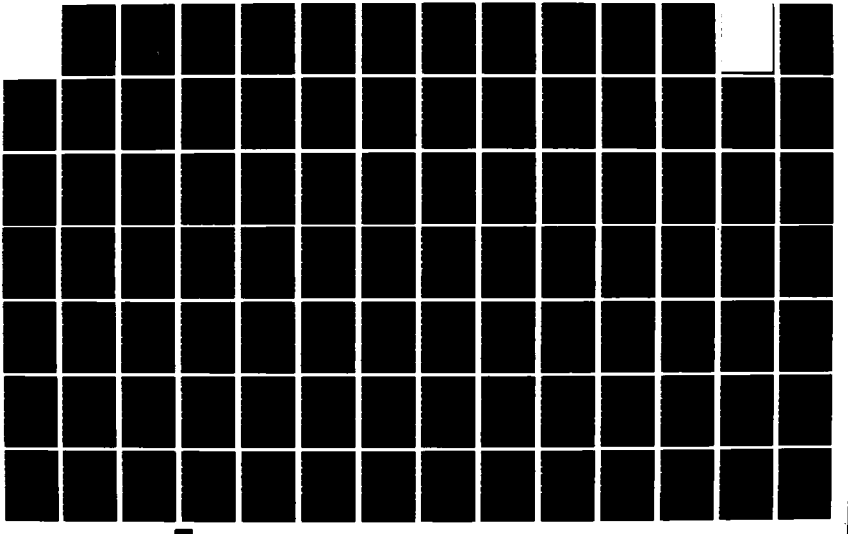
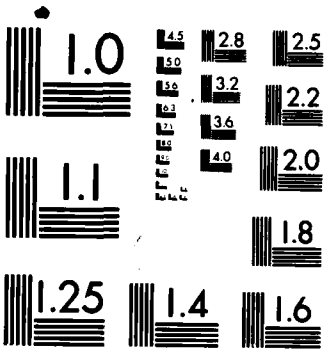
AD-A162 212 DATABASE MANAGEMENT IN DESIGN OPTIMIZATION(U) IOWA UNIV 1/3
IOWA CITY APPLIED-OPTIMAL DESIGN LAB
T SREEKANTAMURTHY ET AL 30 OCT 83 CAD-SS-83-17
UNCLASSIFIED AFOSR-TR-85-1083 AFOSR-82-0322 F/G 5/1 NL





Technical Report No. CAD-SS-83-17

Database Management in Design Optimization

by

T. SreekantaMurthy, S. D. Rajan, C. P. D. Reddy, D. T. Staley,
M. A. Bhatti, and J. S. Arora

DESIGN OPTIMIZATION LABORATORY
Division of Materials Engineering
The University of Iowa
Iowa City, IA 52242

Prepared for
AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
Under Grant No. AFOSR 82-0322

October 1983

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 85-1083	2. GOVT ACCESSION NO. A162 212
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Database Management in Design Optimization	5. TYPE OF REPORT & PERIOD COVERED Annual, October 1982 to September 1983
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) T. SreekantaMurthy, S.D. Rajan, C.P.D. Reddy, D.T. Staley, M.A. Bhatti, J.S. Arora	8. CONTRACT OR GRANT NUMBER(s) AFOSR 82-0322
9. PERFORMING ORGANIZATION NAME AND ADDRESS Division of Material Engineering College of Engineering The University of Iowa, Iowa City, IA 52242	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2307/B1
11. CONTROLLING OFFICE NAME AND ADDRESS AFOSR/NA Building 410 Bolling AFB, D.C. 20332	12. REPORT DATE 30 October 1983
	13. NUMBER OF PAGES 222
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE NA

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Distribution is unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Database Management; Database Design: Optimization; Design; Engineering Computations; Scientific Computing; Interdisciplinary Design; Analysis

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The report describes fundamental concepts, needs, and requirements of a database management system for design optimization. Type of data needing management are identified. A preliminary database management system for structural optimization has been designed, implemented and evaluated. Detailed specifications for a desirable database management system for more general design optimization environment have been developed. Some important specifications for the system are (1) data independence, (2) multiple logical views of

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

the data, (3) memory management, (4) matrix operations utilities, (5) query language for use in interactive sessions as well as applications programs, and (6) management of permanent, temporary, global and local databases. Such capabilities must be available for design optimization applications. A comprehensive review of literature on database management systems for engineering computations has been completed. It is noted that the field is fairly new. Some systems have been developed in the recent past. Their favorable features and limitations are identified. Based on these studies, development of a comprehensive engineering database management system has been in progress. The system is being developed and integrated into design optimization methods. It will become a core for design, implementation and evaluation of databases for engineering optimization applications.

ABSTRACT

The report describes fundamental concepts, needs, and requirements of a database management system for design optimization. Type of data needing management are identified. A preliminary database management system for structural optimization has been designed, implemented and evaluated. Detailed specifications for a desirable database management system for more general design optimization environment have been developed. Some important specifications for the system are (1) data independence, (2) multiple logical views of the data, (3) memory management, (4) matrix operations utilities, (5) query language for use in interactive sessions as well as applications programs, and (6) management of permanent, temporary, global and local databases. Such capabilities must be available for design optimization applications. A comprehensive review of literature on database management systems for engineering computations has been completed. It is noted that the field is fairly new. Some systems have been developed in the recent past. Their favorable features and limitations are identified. Based on these studies, development of a comprehensive engineering database management system has been in progress. The system is being developed and integrated into design optimization methods. It will become a core for design, implementation, and evaluation of databases for engineering optimization applications.

SUMMARY

In October 1982, a project was initiated under the sponsorship of Air Force Office of Scientific Research to study and develop database management concepts for design optimization. Considerable progress has been made since then in identifying the needs of database management concepts, type of data needing management and the features of a database management system suitable for design optimization. Various data models and their suitability for design optimization have been studied. Various file structures for physical data storage have been studied. Suitable structures have been identified. A comprehensive review of literature on database management concepts in scientific and engineering computations has been conducted. Available systems have been evaluated. Favorable features and limitations of the available systems with respect to their usage in the design optimization environment have been identified. A database management system for structural optimization has been designed and evaluated. Based on these studies desirable features of a general purpose database management system have been identified. Such a database management system is badly needed for design development, and evaluation of proper databases for design optimization. The system will become a core for further studies on database development for design optimization.

The report is divided into four parts. A summary of each part, status of research, associated personnel, publications, and plan of research are presented in the following paragraphs.

PART I

Rajan, S.D., Bhatti, M.A. and Arora, J.S., "A Database Management System for Structural Optimization."

This report is partly based on the Ph.D. dissertation of Mr. S.D. Rajan completed in July 1982. The work was in progress when the project was initiated. Following is a summary of this report:

The database management system for design optimization has needs that are characteristic of most engineering computations. However there are some requirements that stretch the limitations of not only computer systems but also the methodology behind software development. This report examines the needs of engineering database management. Details are drawn from the SADDLE system that has been used for optimal design of structural systems. SADDLE stands for Structural Analysis and Dynamic Design LanguagE. The system is modular, portable, reasonably efficient and amenable to future growth. All the modules interact via a global database. The database management system uses a data manager that has three distinct parts — the data model processor, the resource manager and the I/O manager. Both the relational and hierarchical data models are supported. Tuning parameters are provided to enhance computational efficiency on different machines. Emphasis is on user-system interaction — free-format input, error recoveries, and easy means to create, edit and update information. Experience gained with the SADDLE system is utilized in defining a general purpose database management system for design optimization.

Several publications have been planned based on this work.

PART II

SreekantaMurthy, T. and Arora, J.S., "Database Management Concepts in Design Optimization."

This report presents general concepts for database management in design optimization. The need for database management is emphasized. Data used in design optimization is discussed. Various data models used in database management are described. A data model suitable for design optimization is discussed. Physical and logical data structures are studied. File structure for physical data models are important. Therefore various file structures are studied for physical storage of data. Suitable file structures for design optimization are discussed. A comprehensive review of literature for database management in scientific and engineering computations is conducted. It is noted that database management ideas are fairly new to engineering design community. Some database management systems for scientific computing have been developed in the recent literature. Features of these systems are studied. Favorable features as well as drawbacks of the available systems with respect to design optimization applications are noted. Based on this study a suitable database management system and database design for optimization applications can be developed.

The report is a part of the Ph.D. dissertation of Mr. T. SreekantaMurthy that is in progress. Based on this work a paper has been submitted for presentation at the AIAA Conference to be held in May 1984.

In addition, two other Ph.D. students - J.K. Paeng and G.J. Park - are working on applications of design optimization of general dynamic systems using database management concepts. Database of a commercially available finite element program is being changed so that it can be used for design, implementation and evaluation of database concepts with general applications.

PART III

Reddy, C.P.D., SreekantaMurthy, T., Arora, J.S. and Bhatti, M.A., "Engineering Database Management System (EDMS): Concepts and Requirements."

This report describes concepts and requirements of a database management system for engineering design optimization and, in general, scientific computing. Distinction between database management in business and engineering applications is first highlighted. General concepts for design of a database management system in scientific computing and, in particular, engineering design optimization are presented. Based on these concepts and requirements, a set of detailed specifications suitable for database management system (DBMS) has been developed. Such a DBMS can be used in the development, implementation and evaluation of database management concepts and methods for design optimization. Some important specifications for the system are: (1) data independence, (2) multiple logical views of the data, (3) memory management, (4) matrix operation utilities, (5) query language for use in interactive sessions as well as applications programs (useful for

defining optimization problems), and (6) management of permanent, temporary, global and local databases. These capabilities must be present for design optimization applications. Based on the specifications, a database management system called EDMS (Engineering Database Management System), has been initiated. The system is being developed and integrated into design optimization methods. It will be used in the design, implementation and evaluation of database management concepts for design optimization.

Several M.S. and a Ph.D. students are working on this part. C.P.D. Reddy and T. SreekantaMurthy designed and implemented the data definition and data manipulation languages. Scope of these commands is being expanded. D.T. Staley and R. Hotz - two M.S. students - are working on this part. Matrix operations library is being designed. These commands will operate on the data stored in the database and deposit results in the database. These will be extremely useful in developing and evaluating the databases for design optimization. A general purpose query language is being designed. The material will become a M.S. thesis of Mr. R. Hotz.

In addition, two M.S. students - V. Venkatesh and Y.K. Shyy - are working on the design of linear equation and general eigenvalue solvers. Various solution methods and their algorithms are being designed for implementation with the database management system. Such capabilities will be extremely useful for design optimization and other applications. The material will be suitable for M.S. theses of the above two students.

Journal articles are planned based on this work.

PART IV

SreekantaMurthy, T., Reddy, C.P.D., Staley, D.T., Arora, J.S. and Bhatti, M.S., "Engineering Database Management System: EDMS User's Manual."

This report presents a user's manual for the EDMS. The idea here is to show current capabilities of the system. The manual contains various commands for the system. There are two types of commands: file operations and data set operation. File operations define the type of file, opening and closing of a file and other file operations. The data set operation commands include both the data definition as well as data manipulation commands. Limited capability is available as can be seen from the available commands. Other commands for data definition as well as data manipulation are under extensive development. These will be made available soon. Once the system has been developed, it will become a core for study, design, development and evaluation of database design for various optimization applications.

Several M.S. and Ph.D. students have worked on this part of the project. C.P.D. Reddy (M.S.), Don Staley (M.S.), Robert Hotz (M.S.), and T. SreekantaMurthy (Ph.D.) have implemented the commands currently available. C.P.D. Reddy has graduated and left the University. Don Staley will be graduating and leaving at the end of November 1983. Robert Hotz, T. SreekantaMurthy and some new students will continue working on this part of the project under the supervision of the principal investigator. Robert Hotz will be working on the matrix operations library. T. SreekantaMurthy will be working on query language and the data manager.

PREFACE

This report is based on the Ph.D. dissertation of S.D.Rajan completed in the Civil and Environmental Engineering Program of the Division of Materials Engineering. Partial support for the work was provided by the Air Force Office of Scientific Research Grant No.AFOSR-82-0322. In addition the support provided by the Computer-Aided Engineering Laboratory and the Division of Materials Engineering of the College of Engineering is acknowledged.

ABSTRACT

The database management system for design optimization has needs that are characteristic of most engineering computations. However there are some requirements that stretch the limitations of not only computer systems but also the methodology behind software development. This report examines the needs of engineering database management. Details are drawn from the SADDLE system that has been used for optimal design of structural systems. The system is modular, portable, reasonably efficient and amenable to future growth. All the modules interact via a global database. The database management system uses a data manager that has three distinct parts -- the data model processor, the resource manager and the I/O manager. Both the relational and heirarchical data models are supported. Tuning parameters are provided to enhance computational efficiency on different machines. Emphasis is on user-system interaction -- free-format input, error recoveries, and easy means to create, edit and update information.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

ENGINEERING DATABASE MANAGEMENT

1.1 Need for Engineering Database Management

Three developments in the recent past have convinced engineering software developers of the need for centralized (or, unified) database management. First, there is a constant need to expand the capabilities of a program. The flow of logic has reached a stage where all facets of software development - resource allocation, error detection and program documentation - point to the need for an integrated and controlled approach. Second, modular development has shown that in the absence of a common, shared database, more problems are created than solved. Last, it has been recognized that software vendors simply cannot meet the needs of all users. To cater to varied needs, software development must include means to access data by both system and user programmers.

Figure 1.1 illustrates difficulties associated with unchecked program growth (Felippa, 1979). Program components are represented by nodes and connecting paths show the direction of logic and data flow (these paths can be uni- and bi-directional). It is immediately obvious that the architecture is sensitive to the smallest, local modification. A commonly used strategy to restrain this uncontrolled growth is use of the concept and structural hierarchies. Hierarchy (Fig. 1.2) can be applied to software development in two ways : (1) to organize programs into modules, sub-modules, etc., (restricting the connecting paths) and (2) organization of data by refinement of

Figure 1.1 Schematic illustrating program growth

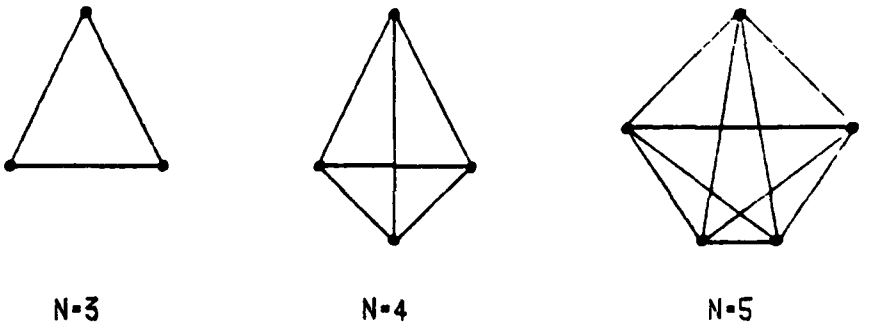
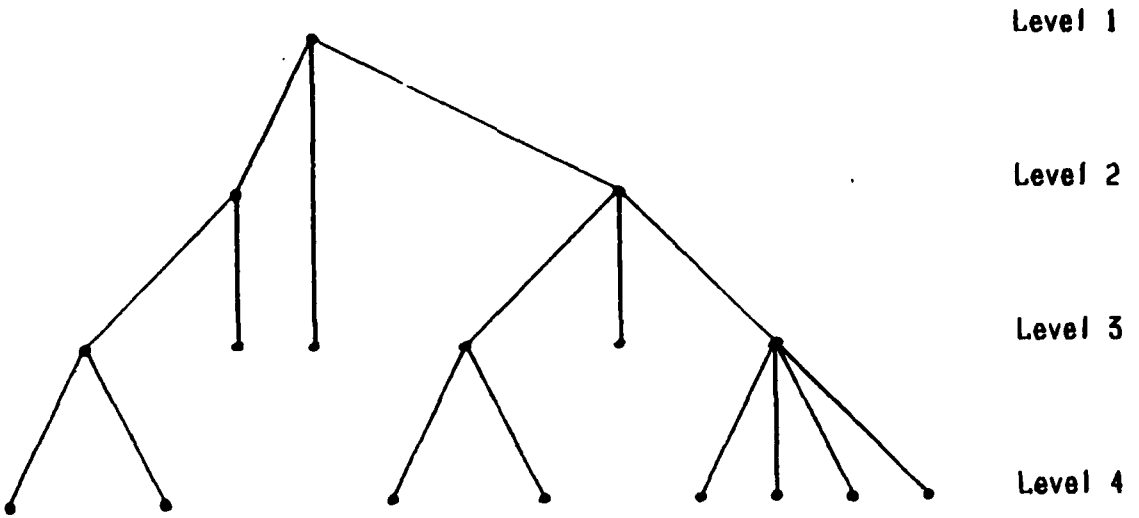


Figure 1.2 The concept of hierarchy



attributes. Modularization provides a means of organizing the program into subdivisions to simplify the growth during the development and enhancement phases. But is modularization alone sufficient ? "Modules are not only characterized by the functions they perform, but also by their connectivity with the rest of the program" (Tausworthe, 1977). The second characteristic forms the basis of centralized data management. Large scale software systems have shown that efficient connectivity can be achieved ad infinitum by coupling modularity with a centralized Data Base Management System (DBMS).

So far, the discussion has been in abstract generalities. In fact, the concepts of database and database management systems have not been detailed. Using Date's (1975) definition, a database is a centralized collection of data, accessible to several application programs and organized according to a database-definition schema. In a design environment, the database will contain information on geometry of the structure (nodal coordinates, element description and connectivity), material properties, loading conditions (element as well as nodal loads), design variables, objective function and similar quantities. The database manager (forming the DBMS) is that part of the program that processes all requests to access, create or update data from the database. It is quite likely that the DBMS will be a part of all modules. Note that generally, neither does the DBMS manipulate data nor does it directly control the flow of logic. These requests are handled by application utilities. This distinction is important due to a common misconception (or, misapprehension) among

engineers. It is believed that a DBMS has very limited scope in engineering computations because it does not manipulate data (equation solver) or process information (graphics package). Another widespread belief is that database managers and database architecture mean the same thing. It is sufficient, at this stage, to compare database managers to equation solvers and database architecture to the generation of global stiffness matrix. As much as equation solvers are inefficient if the global stiffness matrix is formed incoherently (especially true for off-line equation solvers), database managers do a poor job if the architecture is poorly conceived.

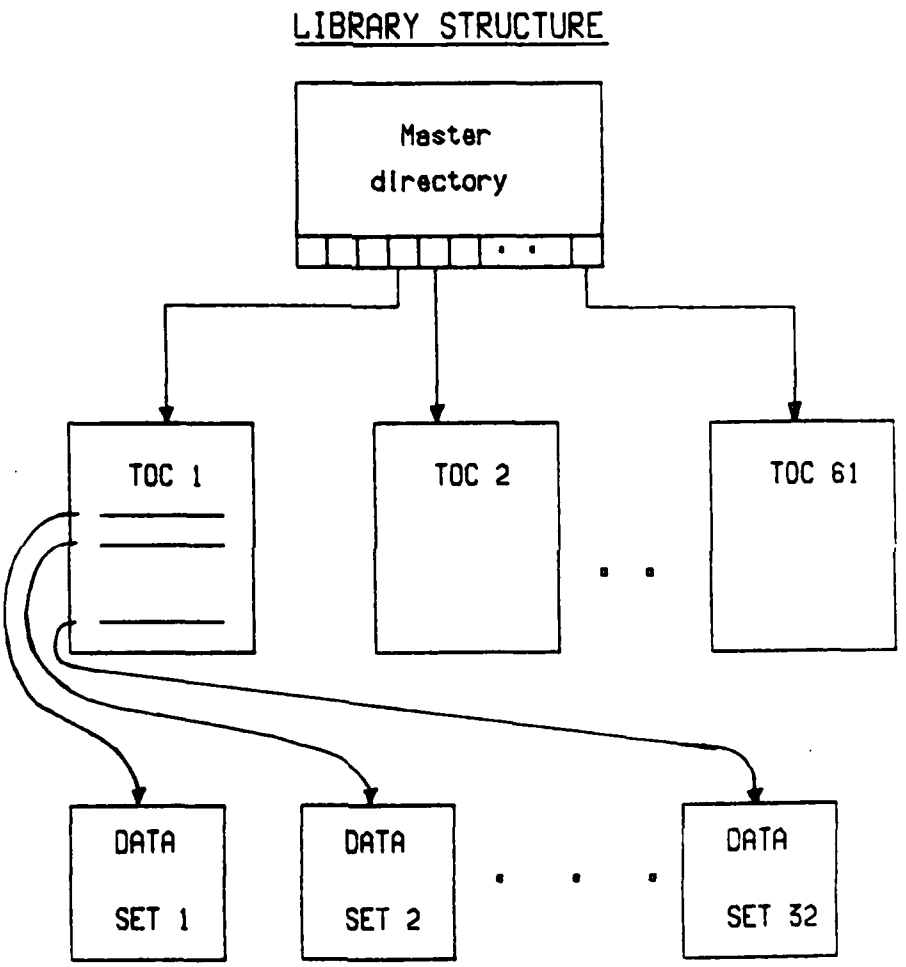
Database management systems of two commonly used finite element software systems, SPAR and GIFTS, are examined next. The SPAR computer software system is a collection of processors that perform particular steps in the finite-element structural analysis procedure (Giles and Haftka, 1978). The data generated by each processor is stored on a database complex that resides on an auxiliary storage device. Each processor has a working storage area that contains the input and the computed data from the processor. Allocation of space in the storage area is problem-dependent and is dynamically allocated during execution. Data transfer takes place directly between a specified location in the working storage area and a specified location on the disk, using a set of data handling utilities. In addition to processors for structural analysis, the SPAR system has processors that form an arithmetic utility system for matrix-related computations, a

data complex utility for managing and printing data and a plotting facility for both on-line and off-line plotting.

The SPAR database complex is composed of a maximum of 26 libraries or data files. The libraries 1 to 20 are available for general use, with libraries 21 to 26 reserved for temporary, internal use. Figure 1.3 shows the structure and contents of a typical SPAR library. The master directory points to the table of contents which in turn points to the data sets referenced in the table of contents. Physically, the auxiliary storage is divided into sectors (of fixed size) and each read/write operation starts at the beginning of a sector. This addressing is a two-word affair -- the first word is the library number and the second word contains the relative sector number within that library. However, a set of data is referred to by giving the library number and a unique name assigned to that data set. At the application level, data sets are accessed by the 'library number-data set name' duo. The procedure to relate data set names to a corresponding relative sector location on disk uses information contained in the table of contents (TOC). Utilities exist in SPAR to store complete libraries as a single data set inside another library, so that a user can organize data in a hierarchy of libraries if necessary. These nested libraries must be reconstituted into separate libraries before the individual data sets can be accessed.

GIFTS is not a single program, but a collection of modules that are present in a program library (Kamel, McCabe and Spector, 1979). Individual modules run independently and communicate via the unified

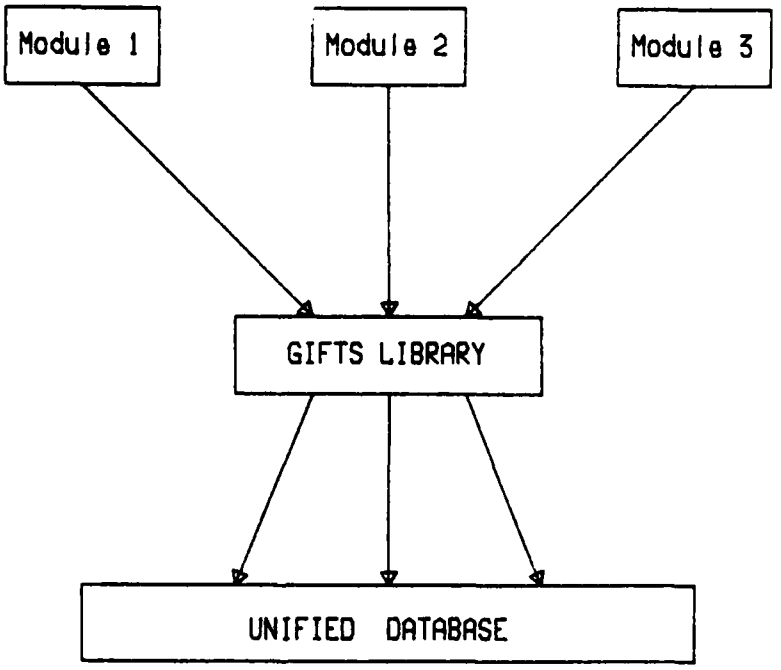
Figure 1.3 Structure and contents of SPAR library



database (Fig. 1.4). The database manager not only processes requests to access data from the database but also does memory management. Each module has its own local database. Working storage is assigned to each data set (in the form of a COMMON block) that is a part of the local database. Each data set is stored in a separate random-access file. The process of paging is carried out within the working storage, unlike a pure virtual memory operating system. There are four routines used with every data set - for opening and initializing the working storage, for reading the data set, for creating/modifying the data set, and for releasing the working storage. Clearly, the GIFTS data management system cannot work in a stand-alone environment since the database manager is embedded in the application software.

The absence of memory management in the program architecture is the principal shortcoming of SPAR. An obvious retort is to let the virtual memory operating system perform this task. There are four reasons why this argument is unsatisfactory. Firstly, supply has always lagged behind demand vis-a-vis memory requirement. The conventional programming approach has been to push the size of arrays as high as possible. Analysis of structural systems with 10000 degrees of freedom and more are no longer uncommon and there are very few machines that can truly offer an in-core solution to these problems. Secondly, all VMOS do not operate efficiently, not because of the lack of methodology as much as the lack of hardware-software interaction that is inherent in most memory management philosophies. Thirdly, with any VMOS, frequent page turning activity can be avoided by localizing references

Figure 1.4 GIFTS modular arrangement



during computation. Once again, the conventional programming approach has not addressed this issue, especially in matrix-type computations. Lastly, distributed processing is bound to become the trend of the future. If computers of varying capabilities are used to access the same database, then quite obviously the problem of resource allocation will be a crucial issue on smaller machines.

There are two other problems with the SPAR data complex. The data definition language promotes a hierarchical database structure; whereas a relational database structure has proven to be physically and logically simpler. The other problem is that excessive fragmentation can take place if the sector size does not happen to be an integral multiple of the data that is stored. Commercialization of the program with its successor EAL has made it even more difficult to find proper documentation on the database structure. Hence, it is quite doubtful whether the program can be adapted efficiently for design purposes.

GIFTS is the state-of-the-art, as far as database architecture is concerned. A relational data structure has been used extensively to store all requisite information. However, the database manager has some severe shortcomings. The primary problem is the size of the data manager. The requirement that four new routines be included with every new data set implies that the size can increase indefinitely. Both programs have a static physical database description, in the sense that minor changes in the structure entail recompilation of all relevant modules. There are elegant techniques available to make file-handling

easier and to manipulate the physical database descriptions. Another problem with GIFTS is that the allocation of primary memory is static. Since different computing systems do not work the same way, a desirable feature is to have tuning parameters externally controlled so that memory can be allocated dynamically. Some of these points are raised in the next two sections.

1.2 Requirements of FEM-based Optimization Software

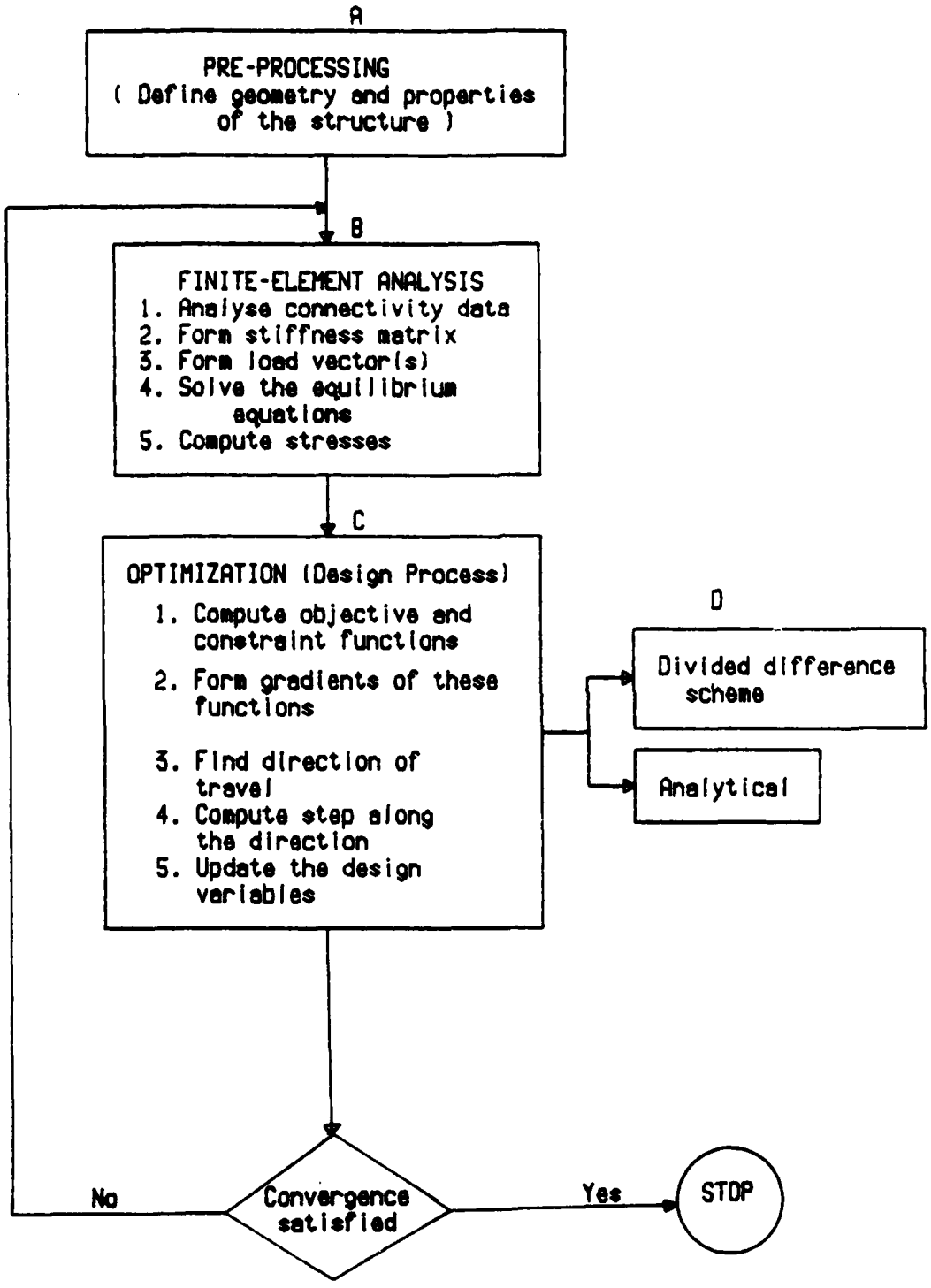
There are significant differences in the flow of a program that performs analysis only and a program dedicated to analysis and design. While finite element techniques are well-established and usually provide solutions to problems in one pass, optimization techniques find solutions in an iterative manner. Figure 1.5 illustrates this fact graphically. The design system calls the analysis program repeatedly and the number of times it is called is dependent on how the gradients of the objective function and the constraints are computed.

Depending on the design philosophy being used, the designer has three options in choosing the design variables:

(i) element cross-sectional properties could be used as design variables; e.g., cross-sectional area, height and width of a rectangular section (Haug and Arora, 1979).

(ii) nodal coordinates could be chosen as design variables; i.e., the shape of the structure is being obtained as the design objective (Haug, Choi, Hou and Yoo, 1981).

Figure 1.5 Flow in a FEM-based optimization program



(iii) element connectivities are the design variables; i.e., the layout of the structure is being determined (Rozvany, 1980).

In a particular design problem, the design variables might belong to any one or all of the above categories.

With the first option, the flow involves performing analysis only once in going from point B to C in Fig. 1.5, if the gradients are provided analytically by the program. However, if a divided difference approximation is used for gradients, design variables are perturbed and an analysis is performed for every perturbation of the design variable. Options (ii) and (iii) introduce further computations, since the pre-processing step needs to be executed whenever the shape of the structure changes appreciably. It is therefore critical to organize not only the computations but also the data in such a manner that the search technique required to locate data from the database is efficient.

The step that requires the most computational effort during analysis is the solution of the equilibrium equations. If data are organized such that the top part of the stiffness matrix reflects the contributions from the 'non-active' part of the structure (the part of the structure that does not change during design), then this part need neither be updated nor decomposed. The savings in overall computation are then substantial. With either option for design variables, the non-active part of the structure grows in size as the optimum design is reached. It should also be mentioned that such a

scheme is used in the analysis of structures with non-linear response (Bhatti, Ciampi, Pister, and Polak, 1981). Furthermore, the decomposed stiffness matrix can be used for finding the solution of adjoint equations for computing the gradients analytically (Haug and Arora, 1979). In addition, computations tend to be repeated if the analysis segment does not compute data required during design calculations. For example, the quantities involved during the formation of the local stiffness matrices are also required for computing the derivatives of the stiffness matrix with respect to design variables.

There is a key difference that has been overlooked so far -- how deep is the user involvement with the development and usage of design software ? The user is usually less inclined to make efforts towards solving problems that are not included among the program development functional objectives (Sobieski, 1980). A "black-box" would be ideal, as in the case of finite element analysis. However, there are two reasons why such a scheme is infeasible with optimization programs:

(i) the objective function and constraints are problem-dependent

(ii) there is insufficient data to conclude which optimization technique works best for structural problems. The user must be given control to select not only the optimization technique but also the values of the 'judgement' (the user must use judgement based on previous experience and intuition) parameters associated with each technique.

It is now clear that a second level of programming effort is involved. The user must be able to clearly define the design

objectives and carry out computations as efficiently as possible. This user effort can be eased by the system programmer. To compute values of the objective function and constraints and their derivatives, the user needs to use certain arithmetic operations and requires the values of the optimization quantities.

1.3 Scope of Work

A computer aided structural analysis and design system, called SADDLE has been developed. This report describes its database management system, which meets most of the requirements outlined in the previous sections. In Chapter 2, the basics of database management are presented. Concepts connected with data structures, physical and conceptual databases are discussed together with discussions on what constitutes a query language. The last section discusses the rudiments of the memory management scheme (MMS). The implementation details in the SADDLE design system are presented in Chapter 3. The relational and hierarchical models together with their file implementation schemes are discussed. Later the SADDLE MMS implementation details are shown. It should be noted that primary storage is finite and is usually a limiting factor. Hence, dynamic allocation of memory is a must for solutions to large problems. The rest of the chapter examines the data structure in detail. Pointer and buffer concepts are introduced, together with data management techniques for vectors and arrays. Information exchange between primary and secondary storage, without the use of I/O buffers, is discussed. Also, the influence of data

structure on computations in virtual memory machines is examined. The chapter concludes with the SADDLE design query language. The evaluation of the system is discussed in Chapter 4. Experience with the use of the SADDLE design system is examined and suggestions for improvements in the features are presented. The report concludes with general discussions on the state-of-the-art in engineering DBMS and scope for future developments in addressing user-system interaction, efficiency and future enhancements of design optimization systems.

CHAPTER 2

ELEMENTS OF A DATABASE

2.1 Introductory Remarks

Between the user, the application program and the computer, there are many levels of abstractions. In the previous chapter, references were made to these different levels - the user view of the database and the conceptual and physical databases. In reality, only the physical database exists, but its implementation is a direct result of the conceptual database. The lowest level is the physical database residing on some secondary storage as bits and bytes. These bits and bytes are based on a conceptual scheme. The conceptual scheme can be described by a data definition language (DDL) that forms the data model. In this chapter, these different levels of abstraction will be closely examined and their role in the SADDLE DBMS will be explained. The presentation is in the order of increasing abstraction. First, the basic problem of storing data will be tackled, followed by the description of the data model in terms of a conceptual language, and finally the user manipulation of data through a 'query language'. In addition, one other closely-related question will be examined - "How to incorporate memory management in the DBMS?"

2.2 Data Structures

A data structure is a structure whose elements are items of data, and whose organization is determined both by the relationships between

the data items and by the access functions that are used to store and retrieve them (Baron and Shapiro, 1980). In this section, some data structures that can be extremely useful for design optimization will be discussed.

Linear Structure : A linear structure is one that is stored in the manner they are processed. These data can then be used to manipulate information such as insertion and deletion of elements of information. Two such structures are described below.

(a) Linear List: This is a finite, ordered list of elements. For example, to store design variable information, it is possible to store the design variable value, the upper bound and the lower bound as fields of an element with each element stored as a record in a file. Another example is a vector that is often used to store information whose size in number of elements is not known a priori or one that cannot be created all at once. In this case, a vector is used as a buffer and the buffer is filled with one element at a time. The buffer is emptied once it is full.

(b) Linked List : In this data structure, each element contains a pointer to its successor. A disadvantage with a linear list is that if elements in the list must be deleted or inserted then either the entire list or a portion of the list must be modified. Only the singly linked list where each element contains a pointer to the next element will be discussed here. The process of insertion is carried out by modification of one pointer - the pointer preceding the new entry (the pointer value of the new entry

now is the same as the old pointer value of the preceding entry). Deletion follows the same process - the pointer value of the preceding entry must be updated.

Trees : A tree is a data structure that is used to represent an hierarchical relationships among data items. A tree is described by nodes; the top level node is the root node. Each node may be connected to a node at the lower level that is the child of the parent node. The discussion on trees is postponed until the next section.

Matrices : These are perhaps the most widely used data structure in engineering programs. They are usually stored in memory in contiguous blocks and are accessed by a simple accessing function. A single-dimensional array is called a vector. Arrays can be two, three and of a higher-orders and are stored either in the column-major order or row-major order. One increasing use of similar data structure is a table where mixed data types are stored. This forms the basis of the relational data model and is discussed later.

2.3 Physical Database

The physical database stores information in a file that consists of records. For example, the database may have a file storing data on all elements of a structure, with each record containing information about one element. These records may or may not have an identical format. The record format links each field to a data type. An operation performed on a file is called a transaction. Typical transactions are:

(a) create/insert a record

(b) delete a record

(c) update a record

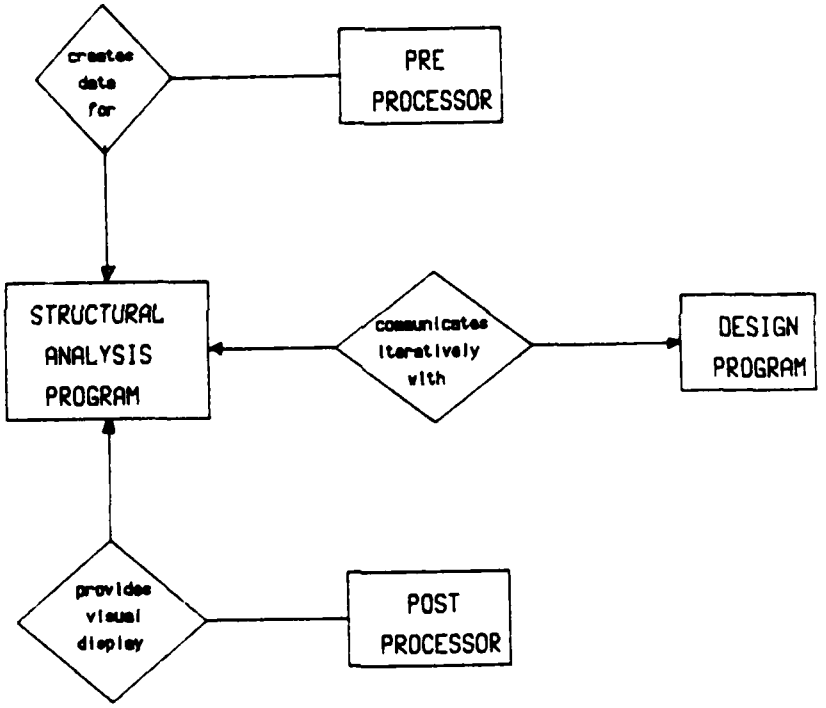
(d) find a record

Note that each transaction is finally performed on one record, though many records may have been used to locate the record in question. The address of a record is usually found through pointers. The concept of pointers will become apparent as the term is used more frequently throughout this chapter.

2.4 Conceptual Database

Before discussing how the physical database is implemented, it is necessary to realize that the information stored in different files is intertwined with the transaction against that information. What this means is that the information cannot be stored arbitrarily, since then its retrieval for future use may be inefficient. Two terms are pertinent in this context-- the entity and the relationship concepts. In Fig. 2.1, the entity, pre-processor is linked to another entity, structural analysis program, by the relationship, "creates data for". Similarly, the structural analysis program "communicates iteratively with" (relationship) the design program (entity) while the pre-processor (entity) "provides visual display" (relationship) for the analysis program. Entities have properties, called attributes, that associate a value from the domain of values for that attribute, with each entity in an entity set. An attribute or a set of attributes

Figure 2.1 Entity-relationship concept



whose values uniquely identify each entity in an entity set is called a key for that entity set.

Entity set	Attributes	Key
Nodes	(nodal coordinates, boundary conditions)	Node Number
Elements	(material, element, element (properties , properties, connectivity)	Element Number

The relationship between entity sets is simply an ordered list of entity sets. In structural applications, two common types of relationship are one-to-one and many-to-one relationships. An example of the former is the correspondence between loading cases and nodal deflections. An example of the latter, is the relationship BELONGS_TO between the entity sets ELEMENTS and MATERIAL_GROUPS (many elements may have the same material properties but no element can belong to more than one material group). These concepts are necessary for implementation of the conceptual database. It is usual to associate an entity set with a file in which the attributes of the entity set are stored as records. Broadly speaking, the files can be divided into four categories :

(a) The size of the file is not known until the user has entered all input. Examples include element information file, nodal data file, and design variable linking information.

(b) The size of the file is computed from user input and until all the requisite information is scanned, the size cannot be determined. A typical example is the stiffness directory

file (a dense index file) in which the directory is created while the element connectivity data is being interpreted.

(c) The size of the file is known and fixed. Examples include the stiffness matrix, loading information and stress files.

(d) The maximum size of the file is known, but the size required during computations is not known. A typical example is the file that stores the derivatives of the active constraints with respect to the design variables. Since the size of the active set is likely to change with each iteration, the current size is always less than or equal to the maximum size.

It is crucial to bear in mind the category of a file before choosing a particular implementation scheme for that file.

There are four widely used implemenation schemes (or, access methods) -- the hashed file organization, the sparse index file organization, the balanced tree organization and the dense index file organization. The idea behind hashing is as follows: The attribute of a pair to be stored or located is used as a parameter for a hash function which is much like the address computation function used for an array. However, unlike array subscripts, which map directly into sequence of integers, attributes to tables are frequently large and are often character strings that do not map into addresses in any methodical way [B1]. In practice, the hashed file is divided into buckets, each of which contains a specified number of records. The hash function h(i) takes as argument a value of the key(i) and produces

an integer from 0 to a maximum value (which is then used as a pointer to the first block). One area in which hashed file organization can be used for structural applications is in the implementation of a matrix directory.

Suppose a program uses a number of files that have alphanumeric identifiers. Let the file names (and their characteristics on one record) be stored in a file with the entries in a random order. In order to obtain the characteristics with the key as the alphanumeric identifier, one could use a hash function h(i) equal to the length of the identifier i modulo 3. This implies there are 3 buckets. Suppose we wish to find the entry, ELEMENTS, in Fig. 2.2 where b_i's are block numbers. Hashing ELEMENTS, we obtain 2 and locate the third entry in the bucket directory, which points to the first block (b_5) for bucket 2. Each non-empty sublock is then searched for entry ELEMENTS. If the record is not found, the header of the block points to the next block. The chaining of block pointers continues until the record is found.

The sparse index file is organized very much like a phone book. The file is sorted based on the key value. A second file is created called the sparse index file that contains a pair (i,b) indicating that the first record in block b has the key i. The index file is then sorted, based on the key value. The strategy to locate a record is to scan the index file until the key i in the file exceeds the required key. The previous block (corresponding to key (i-1)) is scanned completely, until the required record is located (Fig. 2.3). There

Figure 2.2 An example of hashed file organization

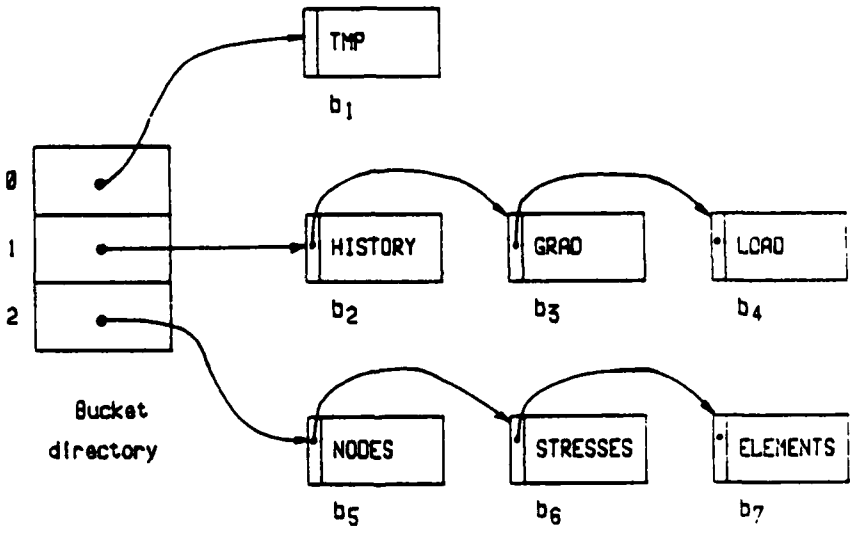
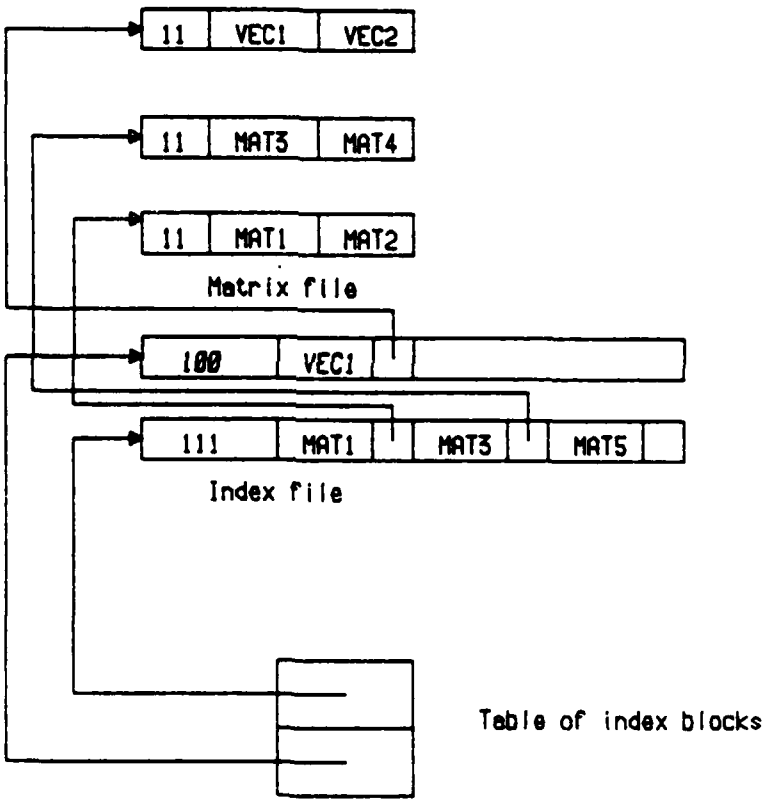


Figure 2.3 An example of sparse index file organization



are different techniques used to scan the block - linear search, binary search or address calculation search. The balanced tree is an extension of the index system. It is sometimes more efficient to have an index of an index, an index of that and so on, until the index fits a block. For most structural problems, these two techniques do not offer considerable advantages over the dense index file organization.

With a category (a) file, in which data fields usually represent attributes, maintaining a sorted file will prove to be expensive. By allowing records to appear in a random order, partially filled blocks can be avoided. This is the idea behind files with a dense index. Insertions can be easily made as the logic involves only keeping track of the last record. The problem with using an unsorted file is to locate a record, given its key. Hence another file (the dense index file) is created, which consists of records (i,b) for each key i in the main file and b is the corresponding pointer to the record in the main file. But should the dense index file exist separately? It should be noted that for a majority of structural applications, the key is an integer value that simplifies calculation of pointers. The dense index could exist as a part of the main file (embedded pointers) - consider a nodal data file in which record j contains the dense index key i. A pointer could exist as a field of the record, pointing to the record containing nodal data for node i. If i equals j, then the record points to itself.

It is not uncommon to have a database store files of differing organizations. It is also not uncommon to associate an entity set

with more than one file, each organized differently. What file implementation works effectively for structural applications can be answered only after the 'data model' is formulated. This will also explain the mechanism to store file types (b), (c) and (d).

At the conceptual level, the data model describes the type of structure used in the database relationships. There are three widely used models - hierarchical (tree-structured), network (plex-structured) and relational (flat files). In all the models, the data definition language is used to tell the DBMS what data structure will be used. Briefly, the functions of the DDL include identification of data subdivisions, establishing the hierarchy in the logical data structure, specification of primary and secondary keys and specification of size in terms of data aggregates of vectors and matrices. The salient features of each model will be illustrated, using an example that is easily understood but somewhat complex - formation of the global stiffness matrix. The hierarchical model is like a tree (Fig. 2.4). The hierarchy consists of elements called nodes. There is a parent-child relationship between elements so that no element can have more than one parent; an element may be related to several elements (children) at a lower level. Figure 2.5 shows a hierarchical data model used to perform the task.

The algorithm is straightforward -

(Step 1) Loop through all elements.

(Step 2) Obtain connectivity information.

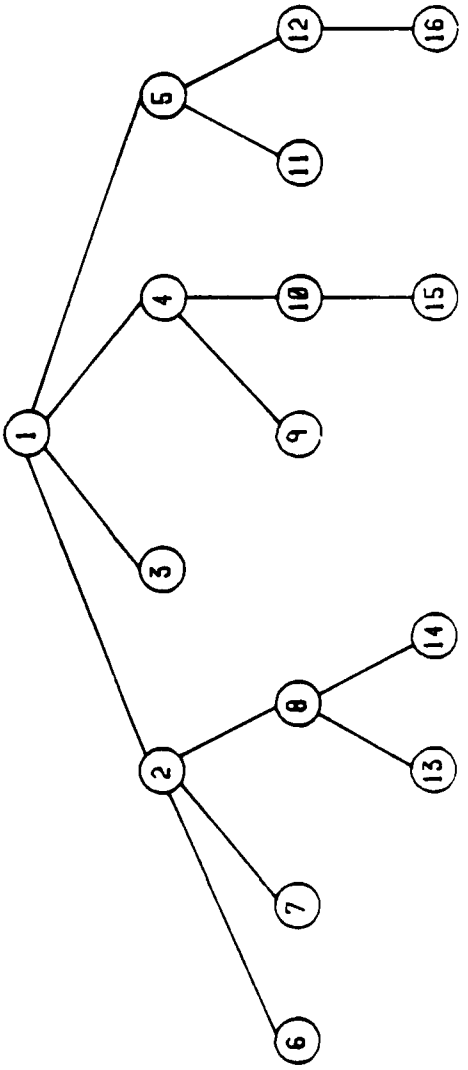
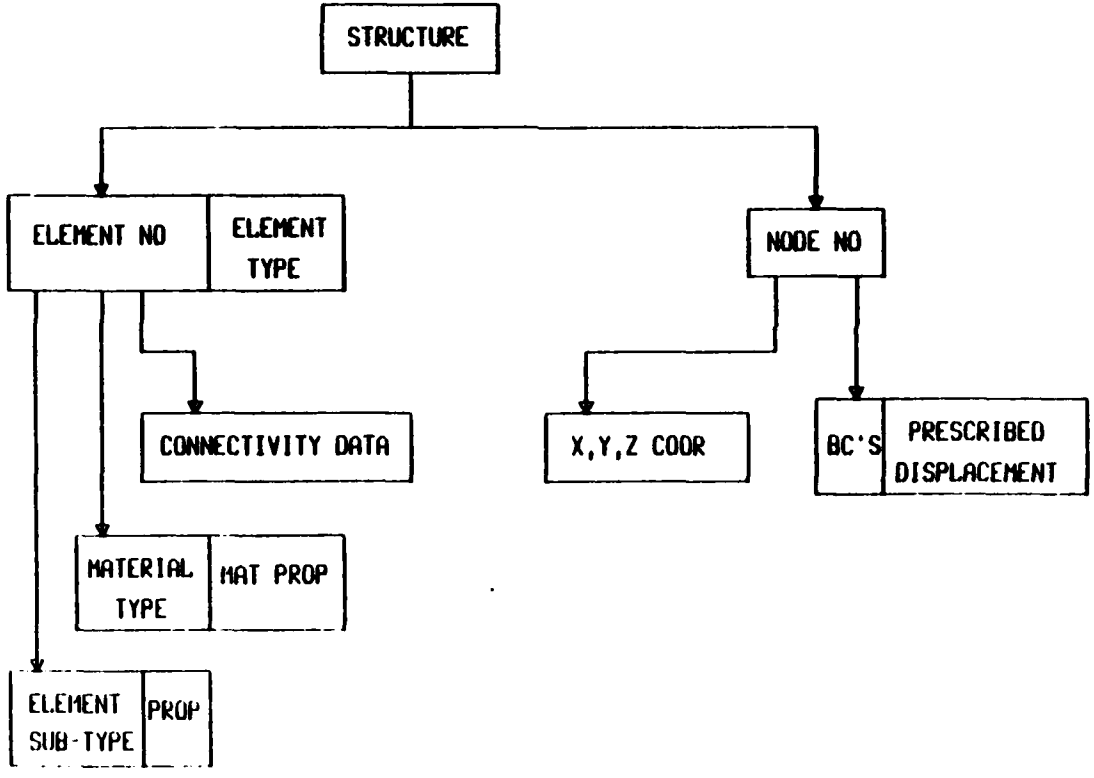


Figure 2.4 Hierarchical database model

Figure 2.5 Hierarchical model for the structural database



(Step 3) Obtain nodal coordinates of the nodes forming the element.

(Step 4) Obtain material properties.

(Step 5) Obtain element properties.

(Step 6) Form the local stiffness matrix.

(Step 7) Form the local-to-global transformation matrix.

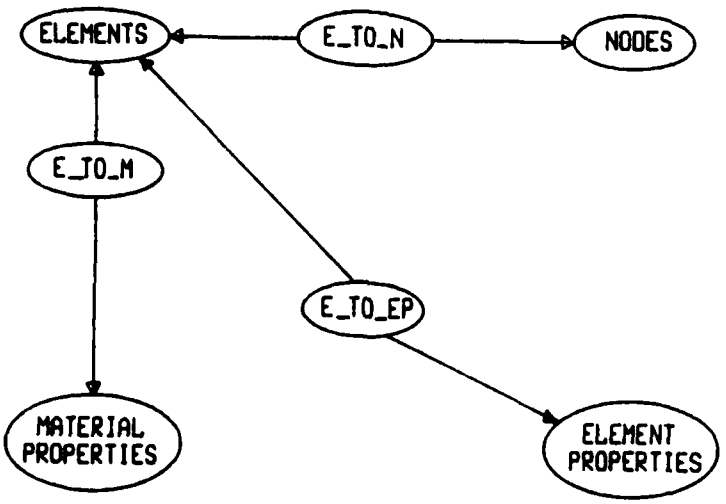
(Step 8) Form the global stiffness matrix, as the product of the matrices in Steps 6 and 7.

(Step 9) Return to Step 1.

In order to implement the algorithm using the heirarchical model, step 2 involves going down the tree to the element CONNECTIVITY. To obtain the nodal coordinates, the CONNECTIVITY record contains pointers to the NODE tree. Similarly, Steps 4 and 5 involve use of MATERIAL_PROPERTIES and ELEMENT_PROPERTIES nodes. The rest of the steps involve computations based on the previously accessed values, with the BOUNDARY_CONDITIONS information used in Step 8.

If the child in a data relationship has more than one parent, then the relationship cannot be described by a heirarchical model. The network data model can instead be used, with all relations restricted to be binary, many-one realtionship (links). To represent relationships among entity sets E1, E2, ... , Ek, a new logical record type T is created as (T1, T2,..., Tk) of entities that are a part of the relationship. This enables the creation of links Li between the entity sets Ei (Ullman, 1980). Figure 2.6 shows a network model for the same algorithm, where a E_TO_M is the logical record type

Figure 2.6 Network model for the structural database

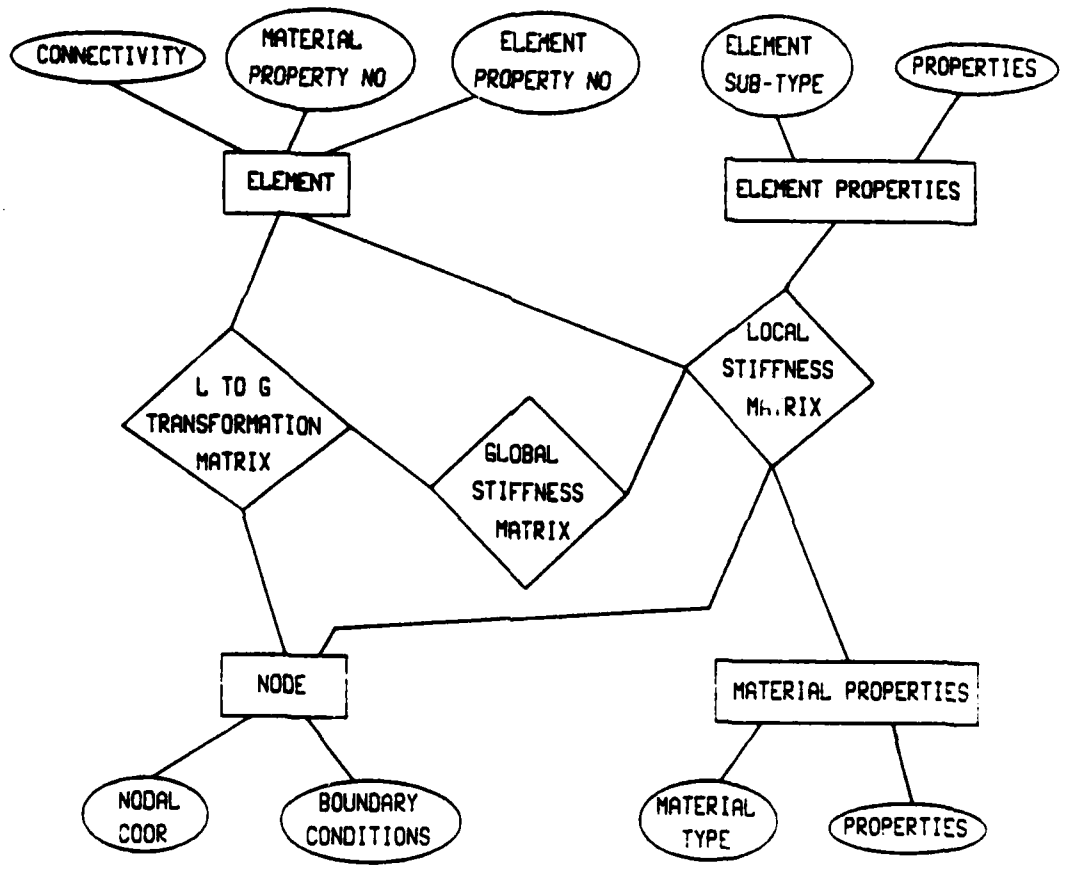


that establishes the many-one relationships between the entity sets, ELEMENTS and MATERIAL_PROPERTIES and so on.

The relational database is based on intuition, tempered by experience, if such a human characteristic exists. With sufficiently diverse databases, both the hierarchical and network models have proven to be functionally inefficient (Date, 1975; Ullman, 1980). Pictorially, the relational scheme is like a table. Each column contains an attribute and each row is an entity. There is a distinct relationship between a row entry and the corresponding column entry and redundant data is not allowed. The mathematical concept of a relational database requires the definitions of a domain, Cartesian product, tuples and a relation. A set of values of one column (attributes) is a domain. If there are n domains, then the Cartesian product of the domains is a set of n-tuples (there are as many tuples as there are columns in the table). A relation scheme is expressed as R(A1,A2,...An) if A1,A2,...An are the attributes of the relation R.

Note the strong similarity between the physical database description in terms of record format, file and record and the relational database description in terms of relation scheme, relation and tuple. This ready translation from concept (relation) to implementation (file), makes the relational database a powerful data model. Figure 2.7 shows the relational scheme for the structural database. The entity set ELEMENTS is related to entity sets MATERIAL_PROPERTIES, ELEMENT_PROPERTIES and NODES. The relation scheme for ELEMENTS has attributes as keys for these entity sets but

Figure 2.7 Relational model for the structural database



not their non-key attributes. Redundant data are avoided and the procedure to link entity sets in a tabular form is called normalization (Codd, 1970).

It is quite apparent that for the structural database in question, the network model is an improvement over the hierarchical model and the relational model provides further improvement over the network model. Note the enormous amount of redundant data in the hierarchical model. The material and element properties are stored for all elements, ignoring the many-to-one relationship alluded to earlier. In fact, the model has already gone through the normalization procedure. In an orthodox model, the tree NODE would have been a child of ELEMENTS. The connectivity data now is of a virtual logical type (which is a pointer to the tree NODE). The network model has minimal redundant data. In fact it can be argued that the model is identical to the relational model, semantics aside. However there is a difference. The logical record types, E_TO_N, E_TO_M and E_TO_EP, are quite unnecessary. In the relational model, they have been substituted by embedded pointers or secondary keys.

Relational models do not provide answers to all situations. The fact that all three models are now in a state of flux indicates that there is still room for improvement. There are instances in which the database relationships, in either the entire or part of the database, are strictly hierarchical, or network or relational. Consider, for example, the SPAR data manager - the hierarchical structure clearly is superior in addressing matrices or submatrices stored as a part of a

huge file. The reason is that the matrix information can be efficiently transferred between the application program and the DBMS and is influenced by the data manipulation language or query language.

2.5 Query Language

The term 'query language' is merely a dignified label for the queries discussed earlier - given an entity set E, its attributes A, and the values of the attributes V, how can relationships be expressed between the quantities (Martin, 1975). In a design environment, queries are usually one of the following forms :

(i) A(E) = ? - "What is the value of attribute A of entity E?" (What is the x-coordinate of node 23?)

(ii) A(?) = V - "What entity E has a value of attribute A equal to V?" (Which nodes have deflections greater than or equal to 2.0 units?)

(iii) A(?) = ? - "What are the values of attribute A for all entities?" (What are the stress values for all elements?)

There are other types of queries that are used less frequently (or, not at all) - ?(E)=V, ?(E)=? or ?(?)=V. Complex queries consisting of combinations of these are also possible - Find the elements in design group 6 that have stress values greater than one-half the yield stress?

Complex queries involve use of multiple keys. The primary key is essentially an entity identifier. The secondary key does not identify

a unique record, but identifies all those that have a certain property. A secondary index uses the secondary key as input and provides a primary key as output, so that a record can be found. The answer to the complex query in the preceding paragraph involves use of primary and secondary keys. The primary key is 6 (design group number) which points to the entity set, ELEMENTS (Fig. 2.8). The entity ELEMENT has two secondary pointers, material group number and stress record pointer, which point the entity sets, MATERIAL_PROPERTIES and STRESSES.

This example is particularly useful in illustrating the fact that the types of queries and the subsequent computations greatly influence the conceptual scheme. Note that the DESIGN_GROUP uses variable length records, since the number of elements in a group is not fixed. The implementation of variable record length files is not trivial. Figure 2.9 illustrates a different entity-relationship scheme. The design group information is now embedded in the entity set ELEMENTS as a data field. The advantage is that the entity set DESIGN_GROUP no longer exists (a variable length record file does not exist). However, in order to answer the same query, the logic would involve scanning the entire file to locate elements in design group 6. Then, is the first scheme superior? The answer is yes, based on the nature of the query. However, if the query is posed differently as -- Find the elements that have stress values greater than one-half the yield stress and create a new entity set STRESS_SUM, which stores the sum of the stress values of all elements in a group, then clearly the

Figure 2.8 Use of primary and secondary keys

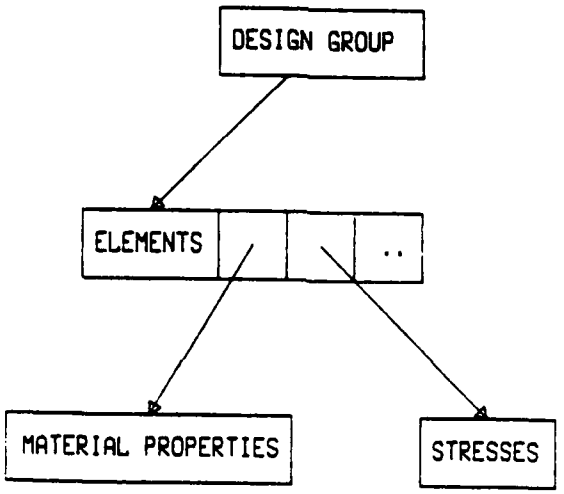
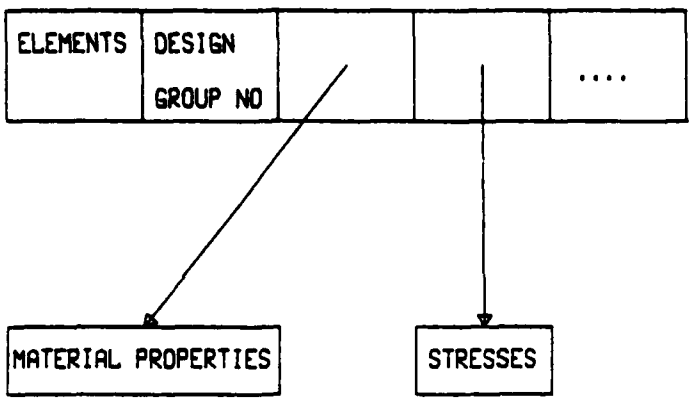


Figure 2.9 Modified relationship scheme of Fig. 2.8



second scheme is superior. What is being emphasized is that the complexity of the query language controls the conceptual database. It should be pointed out that the biggest difference between context-free data managers and application-oriented data managers is the nature of their query language structure.

2.6 Memory Management

The memory management problem is simply this - any computer system has a finite amount of primary memory (core) available for use by the user. If the program requirement exceeds this amount, is a solution to the program possible? There are two types of memory usage - those used by instructions and those used for storing values of arrays (variables). The following discussion is confined to management of array space.

Extending the ideas presented by Shrem (1974) to FEM-based optimization methods, variables can be divided into two types - tabular and matrix.

Tabular	Matrix
Nodal coordinates	Element stiffness matrices
Element topology	Structural stiffness matrix
Loading information	Sensitivity vectors
Design variables	Transformation matrices
Stresses & strains	
Active constraints	
Objective function history	

The working set of data consists of that part of the data that resides in primary memory at a particular stage of the design process (the maximum size of the working set dictates the core requirement). The algorithm should try to keep the working set as constant as possible, in order to minimize use of secondary storage. Quite often, this is not possible. It should also be noted that the ratio between the working sets of tables and matrices changes significantly during the design process, as does the size of the working set itself.

Working Sets in FEM-Based Optimization Techniques

Phase of the design process	Tables	Matrices
Pre-processing	x	
Analysis of input data	x	
Computation of element properties	x	x
Assembly of equilibrium equations	x	x
Solution to equilibrium equations		x
Computation of gradient vectors	x	x
Solution to non-linear programming problem	x	x
Post-processing	x	

The unique nature in which data are required during the design procedure makes it crucial to organize data properly. Consider the example referred to in the previous section. The four quantities (with their abbreviations in parenthesis) that are required are as follows:

(i) connectivity data (CON)

(ii) nodal coordinates (NC)

(iii) material properties (MAT)

(iv) element properties (ELP)

Quantities (i) and (iv) are stored sequentially and accessed sequentially. The size of MAT is usually small compared to other

quantities. The quantity NC is the only quantity whose role in the computations is difficult to predict. If the numbering scheme is random (large half-band width), the node references will be erratic. Hence, a large amount of primary memory should be assigned to NC.

The discussions in the previous two paragraphs provide a basis for allocation of the working set to the quantities involved. If the working set (or the primary memory) is insufficient for the problem being solved, a priority table establishing quotas for the quantities involved is necessary. Since management of the working set is directly under the control of the MMS, it can provide the program an addressable space that is much larger than the physical memory of the computer system. The organization of virtual memory is dependent on the mapping scheme that performs the translation to location space.

One way to carry out the translation is to divide the working set into pages and assign the number of pages to each quantity, depending upon the priority table. In order to implement this scheme, there are three main storage management decisions that must be resolved - fetch, placement and replacement strategies (Shaw, 1974). The fetch strategy defines the policy of when to load virtual memory and how much of virtual memory to load at a time. The placement strategy determines where in the working set to load all or part of the virtual memory. The replacement strategy deals with dynamic allocation systems in which the decision centers around what to remove or swap from the working set when there is not enough space available. The fetch, placement and

replacement strategies used in SADDLE are discussed in the next chapter.

CHAPTER 3

SADDLE : DATABASE MANAGEMENT SYSTEM

3.1 Introductory Remarks

Database management concepts were discussed in Chapter 2. The actual implementation in SADDLE is non-standard and is examined in this chapter. Some additional terminology will be introduced. The management scheme will be dependent on 'file-usage tables' (FUT), 'primary storage directory' (PSD), 'secondary storage directory' (SSD), page table, 'file descriptions' (FDES), assignment and deassignment. The term 'quantity' will denote all the data that are stored in a file. The implementation scheme for the physical database, together with the file-handling conventions, is discussed first followed by the conceptual database. Later, details on the memory management system and the link between primary and secondary storage are explained. Finally, the SADDLE query language structure is explained, followed by the program control structure.

3.2 Software Development

Before developing a design software system, answers to some of the following basic questions are in order (Rajan and Bhatti, 1982):

(a) On what machines will the software be implemented ?

(b) How general-purpose should the program be ?

(c) How familiar are the users (designers) with design systems ?

The answer to the first question is partly dependent on the second. If the design system is to be designed for handling

relatively small problems (of the order of few hundred degrees of freedom and a small number of design variables), then most machines with existing operating systems will be suitable (Kamel, McCabe and DeShazo, 1979). However, if the system is to be more general, then the execution environment becomes restrictive. With the current state-of-the-art in design software, the requirements can be summarized as -

(i) at least a 32-bit machine with double that precision available for floating point representation,

(ii) easy to use file-handling utilities which can be called from within the program, and

(iii) the minimum available core to the user should be at least equal to the module size plus array space of the largest module.

With a *non-virtual machine*, the programmer is aware that there is only limited core available, but secondary storage is unlimited for all practical purposes. With these data, there are three options available to the programmer -

(i) Develop a program in which data is managed entirely within core. Obviously, the maximum size of the problem that can be handled is dependent upon the available core; e.g., MOVIE.BYU (Christiansen, 1980). Usually, restart options cannot be used, since no secondary storage is being used.

(ii) Develop a program with the assumption that a certain fixed core is available and manage the data by storing the quantities on secondary storage and reading them into core only when

required. This method is superior to (i), but since the 'certain amount' is fixed, only a particular class of problems can be handled; e.g., SPAR (Whetstone, 1977).

(iii) Develop a program that does memory management, built along the lines of a virtual memory operating system. This concept permits solutions of considerably larger problems without any changes to the software. In essence, the software is general enough to handle any size practical problem; e.g., GIFTS (Kamel, McCabe and Spector, 1977).

While it is evident that the last option provides the ideal choice for a non-virtual machine, it is not so clear how this scheme peforms under a virtual memory operating system.

The second question has an easier answer. A modular design system allows the program to be as general-purpose as the programmer desires. This is true as long as the database is flexible. Either the initial design of the database makes provisions for future development or the database structure is flexible enough to allow for future development, without involving detailed book-keeping on the part of the system programmer. There are two different approaches to ensure this requirement. The top-down methodology (the hierarchy of development tasks proceeds from a job represented by a node at level n, to jobs represented by its subordinate nodes at level n+1) ensures that the original concepts are always visible. In the bottom-up methodology, one never loses sight of what is being accomplished. Neither method is satisfactory and quite frankly, there is no

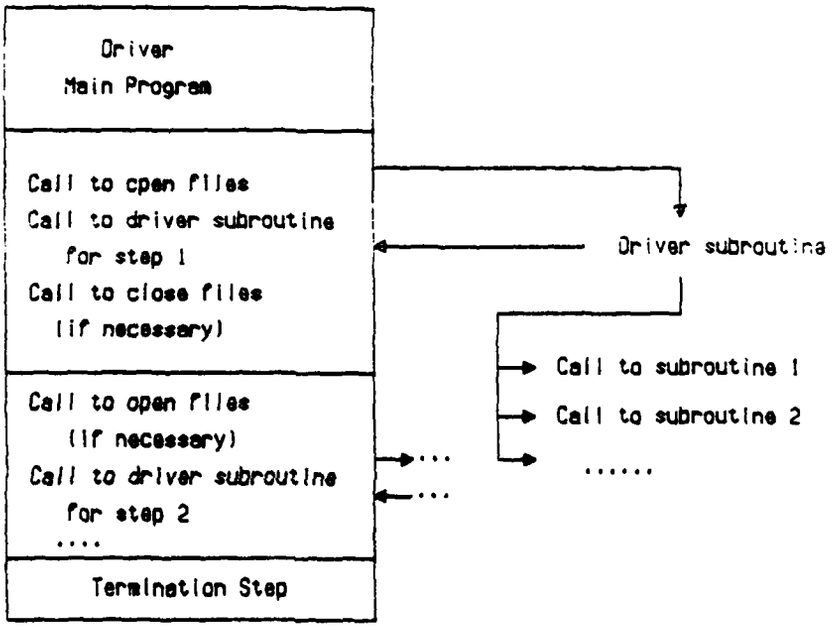
substitute for proper management (software development is still an art).

The last question addresses man-machine interaction. For wide application of the software, it is necessary to evolve a language that can be easily understood by beginners and experienced designers. In this context, the observations of those who used the POL feature of the ICES system is worth examining -- "It is interesting to note that several engineers have found it quite difficult to use problem oriented language. This at first seems a strange paradox since problem oriented language is supposed to simplify computer usage. Closer examination reveals that the users typically are not engineers" (Ross, 1966). The comparison between human languages and computer languages shows that both are far from perfect and extrapolating the fact that the former has taken over 3000 years to evolve to this state, computer languages still have to transcend a significant portion of the evolutionary process. At this stage it seems as if better languages can be developed only through user-feedback.

3.3 SADDLE Physical Database

SADDLE modules communicate with the physical database by reading (and writing) information from the files. File-handling can pose serious book-keeping problems, for several reasons. Figure 3.1 illustrates SADDLE's file-handling conventions. Calls to open and close files are made only in the driver main program. To execute a particular step of the algorithm, quantities involved in the execution

Figure 3.1 File handling conventions



of the step are first identified. The main program then makes calls to open files that contain these quantities (if they are not already open). Control is then transferred to the driver subroutine, which may or may not use calls to other subroutines to execute the logic. Finally, control is transferred back to the driver main program. Before executing subsequent steps, appropriate files are opened/closed.

The SADDLE physical database is made up of two types of files - those that store program control information and those that contain data pertaining to the problem being solved. The files can be subdivided into two types, based on the way data is accessed - sequential access files and random access files. Interaction with sequential access files takes place one record at a time, starting at the beginning of the file. This interaction is relatively standard in FORTRAN. A random access file used in SADDLE is described by its record length and page size. The record length is the number of words of information stored in one physical record and an integral multiple of records make a page. If the size of the file is not known a priori, then the I/O operations are in terms of a single record. If the size of the file is known, then a page of information is transferred between the primary and secondary storage. The CPU time required for the latter case is significantly less. The page size has been set as close the operating system page size as possible. This allows efficient disk operations to take place. There is one subroutine associated with file operations. The subroutine, FILEOP, performs seven operations

associated with file handling - read, write, open, close, delete, check existence of and rename.

Subroutine FILEOP is invoked using six arguments :

SUBROUTINE FILEOP (oper,nunit,fname,ntype,nbuf,np)

oper : operation to be performed. Valid values are -

READ : for reading from a file

WRIT : for writing on a file

OPEN : for opening a file

CLOS : for closing a file

DELE : for deleting a file

EXST : for checking the existence of a file

RENM : for renaming a file

nunit : unit number on which the operation is to be performed/code for checking existence of a file; (nunit=1 if file exists;nunit=-1 if file does not exist)

The unit number is to be supplied by the calling routine for READ, WRIT and CLOS operations. The unit number on which the file is opened is returned for the OPEN operation.

fname : file name

The file name is provided for all operations even though the name is not used for READ,WRIT and CLOS operations, so as to make the program readable.

ntype : file type

Valid values are - SA(sequential access) and RA(random access).

nbuf : buffer used for READ/WRIT operations

A dummy value is specified if the operation is not a READ/WRIT operation. Otherwise the argument corresponds to the array that contains the values to be written on the disk, for a WRIT operation or to the array that stores the values that are read from the disk, or a READ operation.

np : page number/file name

There are two uses of this argument. For a READ/WRIT operation, the value of the page number is supplied by the calling routine. For a special OPEN operation and for the RENM operation, np is used to pass the ordinal value of SPEC and the new name of the file, respectively.

To help facilitate transfer of information between primary and secondary storage, some additional information must be generated. This step is executed by three subroutines; IOCHAN, INITAL and MODEIO. These subroutines carry out I/O definitions and define the entries in the storage directories.

Subroutine IOCHAN does the following :

(a) defines unit numbers for terminal input/output, command input file, printer (output) file and conversation (output) file,

(b) sets the program and database version,

(c) queries the user for job identification and checks for its validity,

(d) sets values of valid file unit numbers.

Subroutine INITAL does the following :

(a) starts timing the execution

(b) reads in file descriptions

(c) initializes program counters

Subroutine MODEIO is used with the command input and save conversation modes of execution. The routine checks for the existence of the file jobname.COM for command input and creates/opens file jobname.OUT for output. If jobname.OUT exists, then the output is appended to the end of the file.

The description of the SADDLE files is given in Appendix B, where the format of a typical record in the file is described.

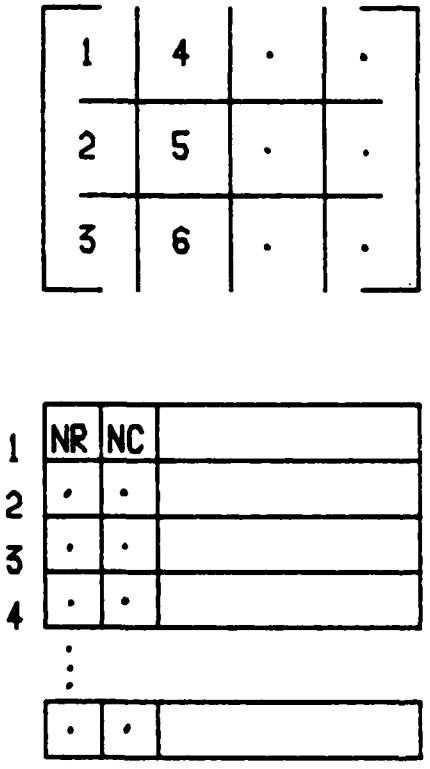
3.4 SADDLE Conceptual Database

Two data models have been used in the SADDLE system. The hierarchical model is used with matrix-type entity sets, while the relational model is used with tabular information. The matrix entity sets can be divided into two categories - sparse matrices and full matrices. An example of a full matrix is the matrix of gradients of active constraints (GRAD) with respect to design variables. All full matrices in SADDLE are stored as submatrices in the column order. Figure 3.2 shows the conceptual arrangement of GRAD. The matrix has NDESV (number of design variables) rows and NAC (number of active constraints) columns. The number of active constraints usually changes from design point to design point, but this does not affect the storage scheme since the matrix is stored column-wise. In order to access data in submatrix (i,j), the record number is computed as,

$$REC = i+(j-1)*NR$$

where NR is the number of superrows in the matrix. In the case of GRAD

Figure 3.2 Storage scheme for entity set GRAD



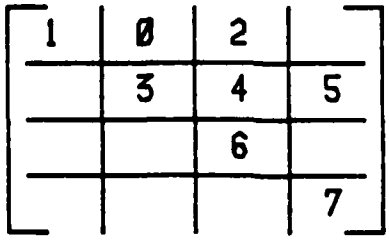
there are NDVB superrows computed as follows -

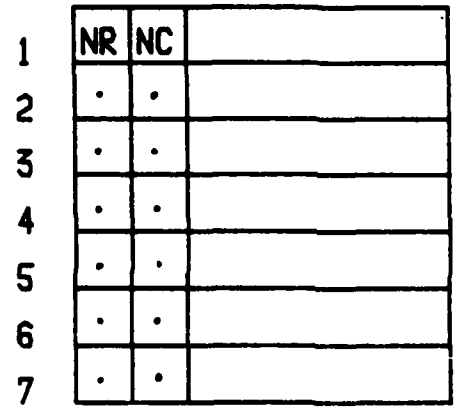
NDVB = NDESV/NBSIZ + MINO(MOD(NDESV,NBSIZ),1)

where the size of one submatrix is (NBSIZ,NBSIZ). This is an example of a sorted file that requires no indexing, since the calculation of REC yields a unique value. Other entity sets that follow a similar conceptual scheme include the constraint function file (FUNC), temporary files used with the linearization technique in module OPTMOD, loading information file (LOAD) and the deflection file (DEFL).

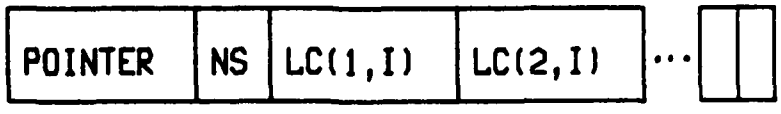
The stiffness matrix (STIF) is an example of a sparse matrix. As in the case of GRAD, the matrix is split into submatrices. However, only the non-zero submatrices in the upper triangle are stored row-wise. Information from STIF can be obtained by reading the location from file, SDIR (Fig. 3.3), a modified sparse index file. The number of records in SDIR is the number of superrows in STIF. The value of POINTER is the record in file STIF, where the first submatrix in that superrow is stored. NS is the number of contiguous strings of non-zero submatrices in that superrow and LC(1,I) is the number of first super-column in string and LC(2,I) is the number of last supercolumn in string. Taking the example in the Fig. 3.3, there are 4 superrows. The first superrow has 2 strings of sub-matrices (LC(1,1)=1, LC(2,1)=1 and LC(1,2)=3, LC(2,2)=3) with the POINTER value 1. Similarly, superrow 2 has one string (LC(1,1)=2, LC(2,1)=4) with the POINTER value 3, and so on. The file STIF contains sorted information, based on the pointer values in SDIR, with the first two fields in each record storing the number of rows and columns in the submatrix.

Figure 3.3 Storage scheme for entity sets SDIR and STIF





STIF



SDIR

The relational model is different in implementation from the hierarchical model, because of the mechanism involved in locating a record of information. The dense index file implementation of two relational entity sets - element information (ELEM) and nodal information (NODE) is now considered. Figure 3.4 shows the conceptual record of the entity set, NODE. The attributes are as follows for the i^{th} record :

NU	user number of system node i
NS	system record where user node i can be found
X,Y,Z	nodal coordinates
DVX,DVY,DVZ	design group numbers of the x, y and z coordinates
DISC(6)	vector of displacement constraint codes (0 : no constraint; 1 : constraint)
NBL	*superrow to which the node belongs in the stiffness* matrix
NFR	row number in the superrow
BC(6)	vector of boundary condition codes (0 : suppressed; 1 : free to move)

The conceptual addressing of a node takes place in two modes - the user mode and the system mode. This scheme is necessary, since (i) the user does not generate information on all the nodes (automatic node/mesh generation) and (ii) the system renumbers the nodes to reduce the half-band width. In order to locate user node j, the j^{th} record in the file is read. If NU=j, then the information has been located. Otherwise, record NS is read, where the user node j is now located. In order to

Figure 3.4 Storage scheme for entity set NODE

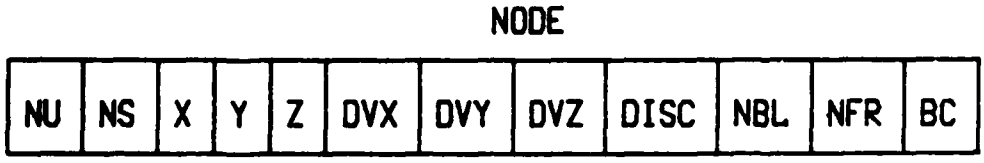
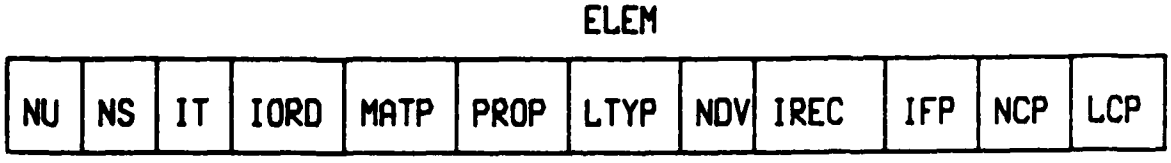


Figure 3.5 Storage scheme for entity set ELEM



locate system node j, the j^{th} record in the file is read. Hence, the embedded pointers NU and NS actually account for the dense index file implementation. Note that the file is arranged so as to minimize access in the system mode (at worst 1 access per node), which is likely to be used more often then the user mode (at worst 2 access per node). The example is further useful in illustrating the fact that the conceptual database can be different than the physical database in implementation. The attributes BC and DISC are binary data and it would be a waste of space to store them as vectors. In actual implementation, they are stored as packed data (into the lowest-order 6 bits of a single variable).

The conceptual record of the entity set ELEM is shown in Fig. 3.5. The attributes are as follows for the i^{th} record :

NU	element user number (NU=$-2^{32}+1$ for a continuation record)
NS	system record number where user element i can be found (in a continuation record, stores rest of the node numbers)
IT	element type (1-ROD, 3-TM3)
IORD	order of element (1-linear, 2-quadratic)
MATP	material property number
PROP	element property number
NDV	number of design variables
LTYP	type of design variable

IREC	design group number of the first design variable in the element
IFP	forward pointer to the next element belonging to the same design group
NCP	number of corner nodes
LCP(27)	list of corner nodes (system mode)

The number of corner nodes varies from finite element to finite element. Reserving the maximum storage locations per element for every element would result in wasted, unused space. In order to minimize this unused space, a conceptual variable record length technique has been implemented, as shown in Fig. 3.5. The first record contains a list of 8 corner nodes. If there are more than 8 nodes per element, information is continued in the next record following the continuation record format. In order to locate information on system element j, the j^{th} record is read. If NU has the value of a continuation record, then the element is not active. If not, the value of NCP is checked. If NCP is greater than 8, then the next record is read and the list of corner nodes is transferred to vector LCP.

A linked list technique is used to handle the problem of storing the design variable linking information. In order to identify all the elements that belong to a design group and related information, the attributes NDV, LTYP, IREC and IFP are used. The first two indicate the number of design variables in the element and the type of design variable (area, moment of inertia, thickness etc.). IREC points to the design group number of the first design variable of the element. IFP

is the pointer to link different elements. Suppose elements 5,10 and 15 belong to the same design group. Then IFP for element 5 is 10, for element 10 is 15 and for element 15 is -1. A negative value indicates that the element is the last element in the group.

Other files that have the same conceptual scheme include the material properties file (MATR), element properties file (PROP) and the element stress file (STRS).

3.5 SADDLE Memory Management System

Drawing an analogy between the physical database and a book, the SADDLE memory management system (MMS) philosophy will be explained. The two entities are compared below.

DATABASE	LITERARY COMPOSITION
Physical Database	Book
File	Chapter
Block	Page
Record	Line
Data	Words

Just as a book is a collection of chapters, the physical database is made up of files. A file (chapter) is a collection os records (lines) grouped together into blocks (pages). However, there are some differences. It would be unusual to find a book with varying page sizes or with the same number of words per line a page. Physical databases usually have different blocking factors and a constant record format for different files.

Using Figs. 3.6 and 3.7, the complex link between the conceptual database, physical database and the primary and secondary storage will

Figure 3.6 Arrangement of primary storage

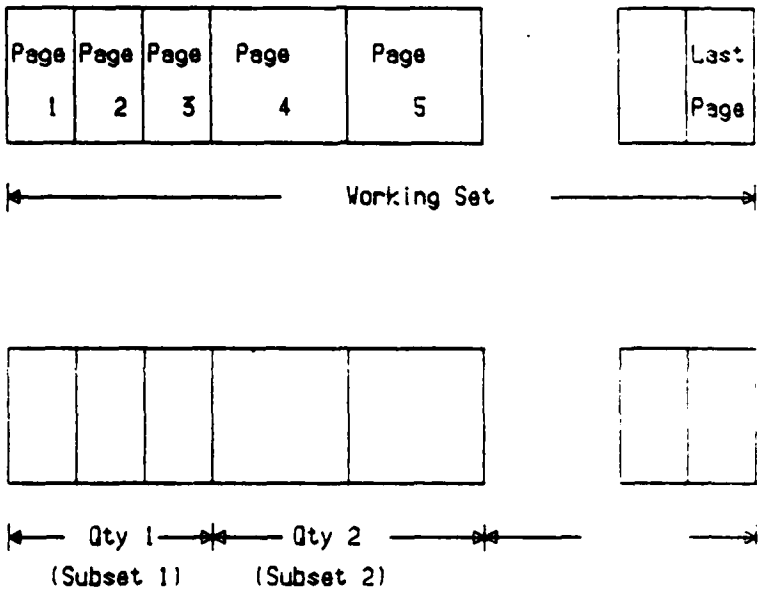
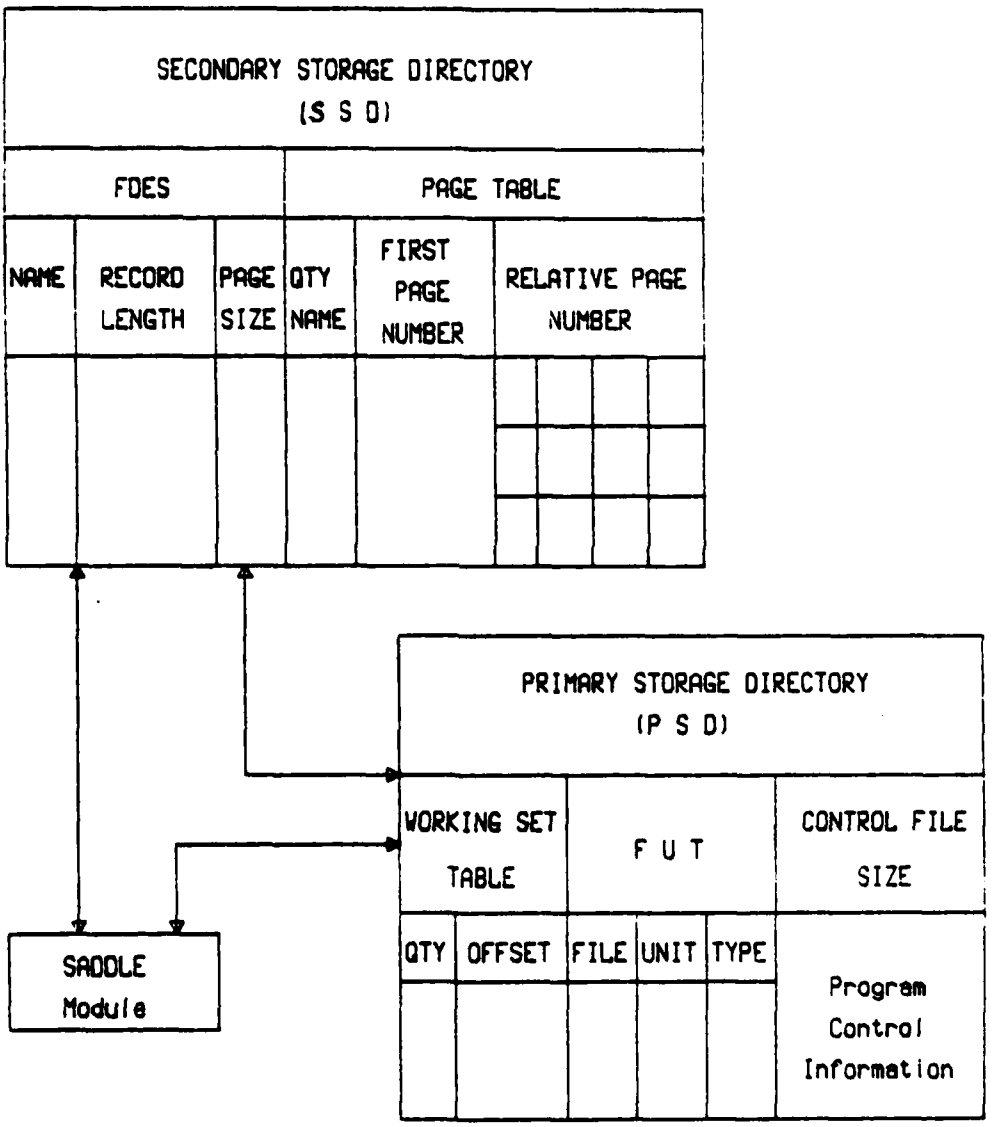


Figure 3.7 Flow between storage directories



be explained. The primary storage is essentially a super-array that is divided into a number of pages. The pages in the working set are each assigned to different quantities and are so arranged that pages that belong to a particular quantity have adjacent locations. The physical database itself, can be looked upon as a collection (super-array) of files (subsets) that have different blocking factors (page sizes). Hence, a unique relationship can be established between the location of a page (in primary storage) and the corresponding location of that page in secondary storage. In order to define this mapping, storage directories have been created in the SADDLE DBMS.

First, file descriptions (name of file, record length, page size) are read from file FDES into the secondary storage directory. Then, a file usage table is created in the primary storage directory. This table records the file name, the unit on which the file is open and the type of file. The control file SIZE is then read. It contains such diverse information as current problem size, problem switch settings and program limitations. Now the assignment of primary storage to each of the quantities can be carried out. This assignment is carried out by routine ASSIGN.

Subroutine ASSIGN is invoked as follows :

SUBROUTINE ASSIGN (KA,name,npage,nofset)

KA : super-array that is the working set

name : name of quantity (file)

This name should exist in the file FDES as a valid entry. Its' file descriptions should be clearly defined in the file FDES.

npage : number of pages to be assigned

The value passed by the calling routine establishes how many pages are to be assigned to the quantity name. If this value is zero, then the maximum number of pages that the quantity requires is assigned.

nofset : offset in the working set

ASSIGN returns to the calling routine the value nofset that is the offset (in terms of locations in KA) from the start of the working set where the subset begins. Routine ASSIGN calls LOAD to load the information from the physical database to the primary storage if the file already exists, or initialize the primary storage if the file is being created. This assignment creates the page table in the secondary storage directory and the working set table in the primary storage directory. The page table has the name of the quantity (hence the file from which data is read/written), the first page number in the working set, and the relative page numbers (of the pages in secondary storage) in the primary storage (relative with respect to the first page). If the relative page number (also called codeword pointer) is zero, then the page is not in the primary storage. The working set table records the name of the quantity and the offset from the beginning of the working set (in number of words of data).

Every request for a transaction follows a set path. Navigation along the path will explain the role of the directories. Suppose the quantity NODE is required in a module. The first step is assignment of space in the working set for NODE. File FDES provides information on the record length and page size of NODE. Then control file SIZE is

used in computing the number of pages required (unless the user specfies a non-zero number of pages for NODE). The page table is updated to reflect the page assignment for NODE. There are two counters associated with every page - the write bit counter and the use bit counter. In addition, a codeword pointer is used with every page. If the pointer value is zero, then the page is not in the working set. Otherwise it points to the location where the page resides. The file NODE is then opened before any transaction is carried out. The unit number is recorded in the file usage table.

Let the transaction require information on a particular record. The routine GETDAT is used for the read operation -

CALL GETDAT (file unit for NODE, record number, working set, KORE value, offset in the working set for NODE).

Using the record number, record length and page size, the page number for the referenced record is computed. The page table is 'looked up' to find the value of the codeword pointer. If the codeword pointer value is zero, the paging algorithm is invoked to make room for the incoming page. Having located the page in the working set, the relative position of the record in the page is computed. The contents are then transferred from the working set to the translating buffer associated with NODE (common block TRNODE). For the write operation, the inverse procedure is carried out (a call to PUTDAT is made).

If a page fault occurs, routine GETPAG implements the replacement strategy. A modified form of the 'near-LRU' (least recently used) technique has been used. Using the secondary storage directory, the

algorithm loops through the pages (in the page table) that are assigned to the quantity in question. The read and write counters are added to determine the page with the lowest sum. This is the page that will be replaced. Before the page is ejected from primary storage, the value of the write counter is checked. A non-zero value indicates that the contents of the page have been modified. The new contents are now dumped on the corresponding page in secondary storage. If all pages have the same sum, then the last page is thrown out. In order to ensure that the new page is not thrown out if a page fault occurs before the page is referenced, the read counter value is set to the highest value of the sum as soon the page is brought in.

Two topics are discussed pertaining to efficiency - tuning parameters and 'garbage collection'. For an efficient design system, the data organization and the memory management schemes must be flexible enough to allow performance enhancements on various computer installations. The SADDLE DBMS includes the following tuning parameters in the design of the database:

(i) maximum size of the working set,

(ii) page allocation policy within the working set, and

(iii) page size.

The first identifiable parameter is the maximum size of the working set. Since the working set requirement is problem dependent, it is advisable to increase this size to as large a figure as possible on non-virtual machines. If the requirements still cannot be met, the MMS will automatically invoke the replacement strategy to handle the

additional requirements. The question is more difficult to answer for virtual machines. The real-to-virtual memory ratio, type of computations carried out and system load (all installation-dependent features) dictate the size of the working set. A smaller working set is preferred for systems in which the machine is heavily loaded. The allocation policy in the distribution of the working set is another parameter. This is especially true since the MMS swaps pages in and out of the subset of the working space that is allocated to the quantity that is being paged. The page size is another tuning parameter. On non-virtual machines, the page size should be close to the maximum I/O transfer allowed by the operating system. On virtual machines, page size should be as close to the system page size as possible. It should be noted that the page size is an important figure. Small page size has the following disadvantages:

(i) more page turning activity, and

(ii) large page table size.

On the other hand, a large page size requires a greater I/O time during paging. The record length could also be considered a tuning parameter, but it is not clear at this stage whether varying the length of the record would yield better results. Paging works most efficiently with programs that execute sequentially, without frequent references to distant memory locations. Programs characterized by such locality of reference generate a minimum of page faults. The SADDLE relational and hierarchical schemes are built on such locality of reference. In order to verify this philosophy, numerical

experiments were carried out. Table 3.1 presents the data obtained from an experiment conducted to compare the SADDLE memory management system against the PRIME operating system.

Numerical experiments were conducted with a structural model consisting of 1250 nodes and 4631 elements. The structure was analyzed for modeling errors such as zero length elements (for line elements) and zero area elements (for two-dimensional elements). The transformation matrix for local-to-global transformation was also computed and design variable groups for the elements were identified. For each of the three tasks, the algorithm proceeded to analyze the data one element at a time, starting with the first element. Three direct access files from the database were used - NODE for nodal information, ELEM for element information and TRAN for transformation matrices, with the last named quantity created and the information associated with the second modified during the course of execution. The numerical results in the table show the direct comparison between an all "in-core" solution (working set 250,000) and a solution with the memory management done by SADDLE MMS. The term "in-core" merely indicates the memory management is being handled by PRIMOS. Both jobs were run simultaneously from different terminals and the system performance was monitered at a third terminal every 10 sec. In Table 3.1, all quantities relating to page sizes and working sets are in terms of 32-bit words, CPU and disk I/O times are in seconds and PF refers to page faults. One fact that is rather clear is that PRIMOS performs relatively poorly when the system is heavily loaded (Run nos.

Table 3.1 Comparison between PRIMOS and SADDLE MMS

Page Size type A

Run No	Working Set Size		Number of Pages for NODE	ELEM	TRAN	Number of SADDLE PF	CPU Time	Disk I/O Time	System PF/sec	% Time CPU Unused
1	21 000		12	20	10	3 933	43.3	57.6	12.7	8.1
	250 000	70	331	91	0	26.4	70.9			
2	65 000	40	60	30	2 153	36.2	59.6	15.7	12.1	
	250 000	70	331	91	0	27.2	95.4			
3	50 000	30	50	20	2 783	38.3	68.4	10.3	11.0	
	250 000	70	331	91	0	26.0	74.5			
4	45 000	30	30	30	2 813	38.9	76.0	15.5	15.1	
	250 000	70	331	91	0	27.1	97.6			
5	80 000	40	80	40	2 103	36.8	85.1	20.4	27.4	
	250 000	70	331	91	0	28.2	108.2			
Page size			486	490	510					

Page Size type B

Run No	Working Set Size		Number of Pages for NODE	ELEM	TRAN	Number of SADDLE PF	CPU Time	Disk I/O Time	System PF/sec	% Time CPU Unused
6	35 000		25	5	5	1 330	30.7	49.7	9.3	22.5
	250 000	35	166	46	0	24.9	60.1			
7	11 000	1	5	5	6 557	60.6	97.2	16.1	24.8	
	250 000	35	166	46	0	26.1	100.9			
8	82 000	10	25	46	2 986	40.8	80.2	13.7	29.6	
	250 000	35	166	46	0	25.4	78.9			
9	85 000	10	50	25	2 957	39.2	56.9	6.1	8.1	
	250 000	35	166	46	0	23.6	51.0			

Table 3.1 (cont)

10	110 000	35	50	25	254	25.0	56.0	13.8	47.0	
	250 000	35	166	46	0	25.0	75.5			
11	70 000	35	25	10	319	25.3	56.5	13.0	33.3	
	250 000	35	166	46	0	24.7	65.7			
12	20 000	10	5	5	3 067	39.2	42.2	6.0	43.0	
	250 000	35	166	46	0	23.	43.9			
13	7 000	5	1	1	3 658	43.3	57.8	6.6	41.0	
	250 000	35	166	46	0	23.4	50.1			
Page size		972	980	1020						

Page Size type C

Run No	Working Set Size	Number of Pages for NODE	ELEM	TRAN	Number of SADDLE PF	CPU Time	Disk I/O Time	System PF/sec	% Time CPU Unused
14	90 000	5	20	5	1 281	38.4	76.2	14.8	21.3
	250 000	12	56	16	0	25.9	85.8		
15	70 000	3	10	10	1 674	42.0	81.4	12.9	8.6
	250 000	12	56	16	0	25.0	79.2		
16	95 000	12	15	5	94	25.2	68.2	22.0	23.8
	250 000	12	56	16	0	26.0	92.5		
17	125 000	12	20	10	79	26.7	102.4	26.5	31.0
	250 000	12	56	16	0	27.5	120.7		
Page size		2916	2940	3060					

5,10,11,16,17). Much smaller working sets fare better, both in terms of CPU time and disk I/O time. However, during times of light system load (Run nos. 1,9,15), PRIMOS required less disk transfer time. Page size types B and C fared much better in terms of number of page faults and disk I/O times, when they are tuned properly, compared to a smaller page size used with type A. The number of pages allocated to a particular quantity also plays an important role in the number of page faults that result when the SADDLE MMS invokes the paging philosophy. This is clear when comparing Run nos. 6 and 14 with Run nos. 8 and 16. The latter show higher page turning activity. Such behaviour is expected since element data are accessed sequentially, while the nodal information references are more erratic. Allocating more pages to nodal data resulted in a smaller number of page faults.

It was mentioned in Chapter 1 that the working set in FEM-based optimization programs changes in nature from one step of the algorithm to another. The implication is that a quantity in use in one step may not be in use in the following step. The question is what to do with the inactive pages. 'Garbage collection' is the process of locating all pages that are no longer in use and adding them to the list of available space. SADDLE does garbage collection only in a few modules. The procedure consistes of two parts. First, the secondary storage directory is scanned, locating the inactive pages. After all inactive pages have been identified, the rest of the pages are linked to form a contiguous block of active pages. The primary and secondary

storage directories are updates to reflect the changes. The procedure is an expensive process and must be used with care.

3.6 Primary-Secondary Storage Link

All information, whether fixed-point or floating-point data (single and double precision), exist as a part of the working set (identified as a super array, KA). When data are initally read from the files, they are stored starting at the location that is the beginning of the subset of the working set allocated to the particular quantity. Similarly, when data are written to the files, they are first transferred to the appropriate locations of the super array before they are actually written on the files. Such a scheme is viable because of the manner in which data are stored in memory. Transfers to and from disks take place in number of words and not in number of variables and data types. Most operating systems make available to the user these low-level I/O routines, which are non-standard FORTRAN routines. They require as input, among other things, a buffer (or,array) that contains the values that are to be written, the location of the buffer in the program space and the number of words that are to be transferred. Such a scheme does not preclude the usage of standard FORTRAN I/O routines, but such a usage would in all probability make the system response poorer, because of the overhead associated with calls to such routines.

SADDLE has five routines that are used to retrieve or dump a particular record from the database. Routines GETDAT and GETREC obtain

information from the working set and transfer the data to the translating buffer. Routines PUTDAT and PUTREC carry out the inverse process of transferring information from the translating buffer to the working set. Routine GETPAG is invoked during a page fault. The idea behind such a scheme is that the implementation of both the relational and heirarchical data models can be carried out with minimal regard to the concept of data types and machine precision.

3.7 SADDLE Design Query Language

The design query language has been designed to cater to the needs of two types of users - the C0 user and the C1 user. The C0 user can use the commands in the SADDLE design language to manipulate data in the physical database. These commands can be divided into four categories. A Category 1 command is a technical information command of the form

(a) HIST
1,100

(b) MENU
ROD2

where the first line of input (alphanumeric) is an entity set identifier and the second line specifies the domain of the entity set (tuples or attribute domain). With the HIST command, the entity set 'Optimization History' is identified and the input '1,100' specifies the range of values of the primary key, design cycle number. If the number of design cycles is less than or equal to 100, information on the entire set has been requested. With the MENU command, the entity set 'Elements' is identified. However, the second line of input ROD2

specifies that all possible values of the attribute, design variable type, be displayed.

The second category command is an update command; information is created, modified or deleted,

(a) SETB,2
1,2
3,4

(b) DESELEM
5

The first line is an entity set identifier. The second (and subsequent lines) of input are either values of attributes or values of primary key. With the SETB command, the entity set 'Design Variables' and its attributes lower/upper bounds are identified. The values '1,2' and '3,4' establish the lower/upper bounds of the current design variables. The command DESELEM cuts across entity set boundaries. The command links the information between the entity set 'Elements', 'Element Properties' and 'Design Variables'. All elements that have the same element properties as element 5 belong to the current design variable group. Element 5 is the primary entity with its attribute, element property number, as the secondary key. The values of the element properties (that are design variables) are transferred to the attribute, design variable value, of the entity, (current) design variable.

Category 3 commands deal with program control information,

(a) CYCLES
20

(b) STATUS

The command CYCLES sets the maximum number of design cycles to 20.

STATUS displays the design information, number of design variables, inequality and equality constraints, design cycles and maximum anticipated number of active constraints.

Miscellaneous commands form the category 4 commands. They include commands to identify types of design variables (element properties, nodal coordinates), optimization technique in use, breakpoint value, value of perturbation used in finite differences, etc.

The CI user has more control over the types of queries. However, the relational calculus that is used to access both relational and hierarchical data, must be programmed by the user. Essentially, the OPTQTY library provides the user with the means to access and update information. The access mechanism operates on one conceptual record at a time either providing access to the entire tuple (ZSTRS) or one or more attributes forming the tuple (ZNODE).

3.8 Control Structure

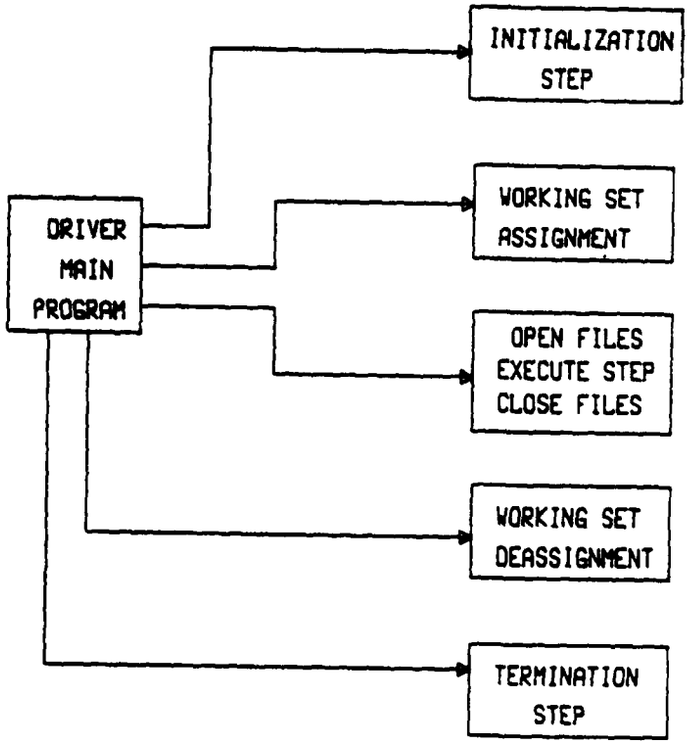
Execution of as SADDLE module can be divided into five phases (Fig. 3.8). The five phases are initiated by a driver main program, whose functions are to:

(i) set working set size (value of KORE). The value of KORE is dependent upon the installation and the operating system,

(ii) allocate subsets of the working space to the quantities involved,

(iii) transfer control to the user command input parser that tests the user input for validity and then transfers control to

Figure 3.8 Flow in a typical module



the appropriate subroutines,

(iv) release the working space (either at every stage of execution or at the conclusion of the execution), and

(v) terminate execution and print user information messages.

Phase 1 : Initiation of Execution

Before execution begins, the SADDLE system performs some perliminary tasks. First, it establishes I/O channels in order to transfer data between the user and the physical database. It then queries the user for a job identification (id). Having established the job id, it checks for existence of such a job in the user disk space. It also makes sure that the user is using the current version of the program (that the databases are comptabile). The read/write counters are initialized and the program counters are set. The working set size is also established as a part of the primary storage directory. The file characteristics are read from file FDES and are stored as a part of the secondary storage direcrtory.

Phase 2 : Assignment of Working Space

The working space is divided into pages, usually of unequal sizes but close to the value that allows optimum I/O transfers. The allocation of pages is left to the user, through the use of reset controls. The user needs to reset the values only if the user feels that either the problem size is larger than the current size of the working set, or that SADDLE system does a better memory management job than the operating system. For non-virtual machines or for virtual machines with small virtual-to-real ratios, SADDLE's paging scheme will have to

be used to solve large problems. Normally, when reset controls are not used, SADDLE assigns the maximum number of pages needed by a particular quantity. The maximum number of pages is automatically calculated by the program (subroutine INITAL in the SADDLE library) during Phase 1 of execution. The user should reset values only if the user is conversant with the quantities involved in the computations.

Phase 3 : Execution Phase

Step 1 of Algorithm:

The driver main program now transfers control to the appropriate subroutine. It passes the entire working set and the values of the offsets of the quantities from the beginning of the working set as the arguments. The computations involved in this phase are carried out and the values from the database are fetched from the database are fetched by appropriate calls to the SADDLE library routines. When computations are complete, control is returned to the driver main program, which then initiates execution of the next phase of the algorithm.

Step 2 of Algorithm:

Step ....

Step ...

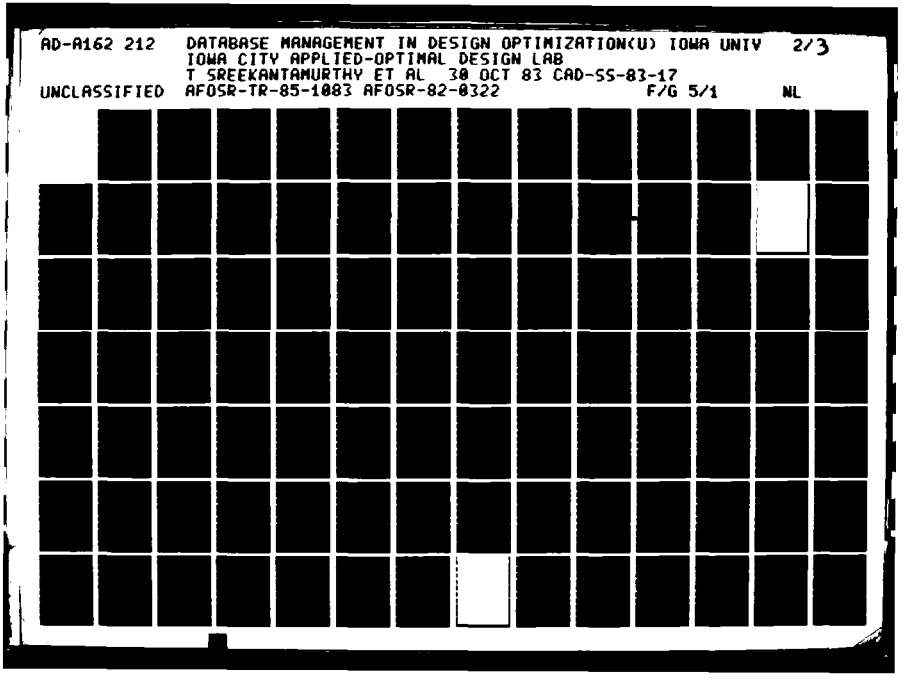
Phase 4: Deassignment of Working Space

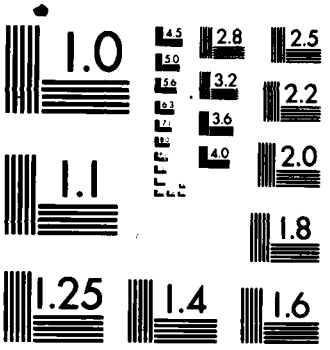
When all of the computations are complete, the SADDLE system performs some more computations - the deassignment phase is exactly the opposite of the assignment phase. Each quantity previously assigned to the subsets of the working space must now be released, but before

the space is made available, the SADDLE system checks whether the quantity was either created or modified during execution of the module. If a page was created or modified, then the new value is dumped on the files.

Phase 5: Termination of Execution

To provide the user with meaningful information (viz. CPU time, paging time, I/O operations), the SADDLE system makes available to the user these important values. On subsequent executions, the user is most likely to use the interpretations of the previous run so as to make the execution more efficient.


AD-A162 212 DATABASE MANAGEMENT IN DESIGN OPTIMIZATION(U) IOWA UNIV 2/3
IOWA CITY APPLIED-OPTIMAL DESIGN LAB
T SREEKANTAMURTHY ET AL 30 OCT 83 CAD-SS-83-17
UNCLASSIFIED AFOSR-TR-85-1083 AFOSR-82-0322 F/G 5/1 NL



CHAPTER 4

EVALUATION OF SADDLE DBMS

4.1 Introductory Remarks

This chapter summarizes the experience with the use of SADDLE system for design optimization. The system has been used for optimal design of various structures using the techniques connected with finite element analysis, non-linear programming, computer graphics and database management systems. The system has access to powerful pre- and post-processors, GIFTS and MOVIE.BYU. The chapter concludes with an evaluation of the SADDLE DBMS and areas of future research work that will enhance the capabilities of the system are outlined.

4.2 Experience with SADDLE DBMS

The design system has been used to solve two different types of optimal design problems - minimum weight design problems and min-max problems. For these design problems, the design variables have been of three types - member properties, nodal coordinates and non-structural (dummy) design variable. Different constraints have been imposed - direct stress and buckling constraint (for truss elements), von Mises stress criterion (for constant stress triangular elements) and displacement constraints at specified locations. Before proceeding any further, the requirements of a typical design problem will be examined. The design problem is usually of the form -

minimize objective function (a function of design variables)

subject to constraint functions ≤ 0 (functions of design variables).

The tacit assumption is that both the objective function and the constraints can be reduced to the above form. Quite often in structural applications, the objective and constraint functions are dependent on other variables, e.g. state variables. One of several techniques (direct differentiation, adjoint variable technique) must then be used by the user to achieve the task of reducing the design problem to the above form. In order to find the next design point, a typical NLP technique will require these four quantities -

(1) value of objective function,

(2) value(s) of constraint(s),

(3) gradient of objective function with respect to the design variables, and

(4) gradient(s) of the constraint(s) with respect to the design variables, for the current design point.

The implication so far has been that the designer(user) must be allowed access to not only data already computed by the rest of the design system, but must be allowed to (i) modify them if necessary, to reflect the design problem based on the user's requirements, and (ii) store/retrieve user-defined data in a similar manner.

The SADDLE design system has provided most of these capabilities for both the C1 and the C2 users. The utility libraries (AOL and OPTQTY) provide access to the database to retrieve, manipulate and update system-computed data. Most of the design information is

automatically generated by module DATUM, where the user can interactively input and edit design-related information. The system also provides space to store user-defined variables. Such a capability is necessary since the user-supplied DESIGN module controls the flow between the analyzer and the synthesizer and in order to be able to achieve this result, the user must be allowed means to manipulate some (flow) control variables.

The following are some other charactersitics (with particular reference to the DBMS) of the SADDLE system -

(1) Supports both the hierarchical and relational data models. The data manager treats both these data models in a similar fashion so that the data transfer between the routines that implement the conceptual database and the routines that implement the physical database is not bound by the data model.

(2) Implements the virtual memory management philosophy. The application program is not bound by the amount of core memory available. The size of the problem that can be handled is dependent only on the size of the page table and the amount of secondary table available.

(3) Caters to the need of different types of users. The user involvement with the details of the data manipulation language is left to the user.

(4) Tuning parameters are provided to enhance the computational efficiency on different machines. Locality of reference is built into the database structure so that minimum page faults are

generated when data are manipulated.

(5) Provides easy means to enhance the capabilities of the system. The data independence that is also alluded to here is not total but changing the database structure of any entity set will involve minimal changes to the application program.

4.3 Improvements of SADDLE DBMS

If there is one major problem with the curent developments in the area of engineering database management systems, it is with the basic concepts and notions connected with the definition of what constitutes a DBMS. Unlike business computing, where the user development of so-called business application software, is rather limited, engineering computing thrives on software development. This trend is based on the premise that individual requirements are peculiar and deserves special attention. The development of engineering DBMS usually proceeds hand-in-hand with the development of application software. The development of the SADDLE system is a typical example. GIFTS, SPAR and SADDLE, in that order, represent an attempt to separate the application software from the components of a DBMS - the data definition language and the data manipulation language. The question is not whether the components can be separated (they have traditionally been distinct from application software) but whether such a separation will aid engineering computations. Specifically, can such a philosophy -

(1) standardize software development, make it easier to control current development, enhance the capabilities of and achieve a

reasonable reliability in usage,

(2) increase computational efficiency and leave the problem of resource allocation to be sorted by the DBMS rather than the system programmer, and

(3) make it easy for both the system programmer and the end-user to form the conceptual scheme of the application software and manipulate the data using the conceptual definitions.

The answer to each of the question is 'yes'. In order to implement such a philosophy into the SADDLE DBMS and enhance the capabilities of the system, the following are some of the areas of future research work-

(1) Build a query language into the system that can be used both interactively and through an application program. This language should make it possible for the user to specify to the system the (conceptual) schema. The system then should be able to decide :

   (a) The data model that is best suited for the (conceptual) schema.

   (b) Form the physical format for the entity sets minimizing redundancy and building integrity into the system.

   (c) Identify the file implementation scheme best suited to the entity set in question.

(2) The query language should be easy to use and some form of query optimization must take place to minimize execution time for certain queries.

(3) The data manager should be able to handle multiple-user requests, implement read/write locks and have a procedure to protect the security of the system.

(4) The system should be able to recover from a system crash. If a total recovery is not possible, then checks and bounds must be built into the system to warn the user of any 'bad' data.

The suggestions outlined above may be a tall order to fill in a short period of time. However, they are very useful additions that can be made to the DBMS over a period of time. The next chapter looks at the application aspects of such a database management system.

APPENDIX A

A brief definition of some important terms are given below. It should be noted that these definitions are not universal, but clarify the usage in the context of this dissertation.

attributes

the properties of an entity

command file

a file that contains commands (and/or data) in the same form as the input the user types at the terminal

data manager

the part of the DBMS that processes requests from the application program and operates on the physical database

data model

a description of the type of structure used in database relationships

database

collection of data (in an organized form) needed for the execution of a module

entity set

collection of entities. Items about which information is stored is an entity.

file

an organized collection of related data records

garbage collection

the process of locating all pages that are no longer in use and adding them to the list of available space.

interactive mode

a conversation mode between SADDLE and the user where the user responses are typed at the terminal

library

a collection of subroutines that perform primitive functions like file operations, input/output functions, graphics etc.

memory management system

a system that oversees the memory allocation to the different entity sets in the program and makes it appear as if more memory is available than what the computer actually has

module

a program that performs an identifiable task

page

a basic unit of primary storage; also basic transaction unit between primary and secondary storage

post-processing

program(s) engaged in the task of helping the user visually interpret output from the analysis and optimization phases

pre-processing

program(s) engaged in the task of preparing input data for the (finite-element) analysis and optimization phases

primary storage

the main storage (core) in the computer

processor

same as a module

pseudo-batch mode

a conversation mode between SADDLE and the user where the user responses are being read from the command file as opposed to the terminal

record

smallest data field that holds a quantum of information (an integral fraction of the page size)

runstream mode

an automated mode of execution where modules are executed in the proper sequence without interactive user intervention; the user communicates with the system through a runstream file

secondary storage

storage on a secondary medium like disk drive or magnetic tape

storage directories

a collection of information on the physical descriptions of the files existing as a part of the database

virtual machines

a machine using an operating system that allows more storage space than the physical size of the memory

working set

superset formed by the collection of all the entity sets used by the module

APPENDIX B

The following is a description of the SADDLE database. It contains a description of the contents of a typical record in every file used by the SADDLE system.

****

FILE NAME : job . SIZE

RECORD LENGTH : 25 INTEGERS
RECORD : 1

VARIABLE NAME	FUNCTION
1	Unused
2	Unused
3	Unused
4 NRUXM	Number of raster units in x-direction on main viewing area
5 NRUYM	Number of raster units in y-direction on main viewing area
6 NRUXM	Number of raster units in x-direction on offset viewing area
7 NRUYM	Number of raster units in y-direction on offset viewing area
8-25	Unused

RECORD : 2

VARIABLE NAME	FUNCTION
1 NCPPEM	Max. number of corner points per element
2 NFGM	Max. number of freedoms per group (block size)
3 NSRM	Max. number of submatrix strings in a superrow
4 NLCSM	Max. number of loading cases / Max. number of fictitious load vectors that can be solved simultaneously
5 NSTRM	Max. number of stresses per stress point
6 NACLIM	Max. number of active constraints
7 NDVPEM	Max. number of design variables per element
8-25	Unused

RECORD : 3

	VARIABLE NAME	FUNCTION
1	NGPT	Total number of points (active+deleted)
2	NGPA	Total number of active points
3	NELTS	Total number of elements (active+deleted)
4	NKPT	Total number of key points (not active)
5	NL	Total number of lines (not active)
6	NG	Total number of surface grids (not active)
7	NS	Total number of solid grids (not active)
8	NMATR	Total number of material properties
9	NPROP	Total number of element properties
10	NDFPM	Max. number of degrees of freedom per node
11		Unused
12	NUGRP	Total number of unknown groups in solution
13	NLCT	Total number of loading cases
14	NLCA	Total number of active loading cases
15		Unused
16		Unused
17		Unused
18	NSTRR	Total number of records in stress file
19		Unused
20		Unused
21		Unused
22	NFIB	Block number of first freedom to be condensed
23	NUNKT	Total number of degrees of freedom in structure
24		Unused
25		Unused

RECORD : 4

	VARIABLE NAME	FUNCTION
1	ISTPRE	Finite element model generation switch
2	ISTPM	Point mass generation switch
3	ISTBWO	Bandwidth optimization switch
4	ISTST	Stiffness formation switch
5		Unused
6	ISTLD	Load generation switch
7	ISTBC	Boundary condition switch
8	ISTDEC	Stiffness decomposition switch
9	ISTDN	Deflection generation switch
10		Unused
11		Unused
12		Unused

13	ISTES	Element stress switch
14-17		Unused
18	ISDES	Design information switch
19-25		Unused

RECORD : 5 UNUSED

RECORD : 6

VARIABLE	NAME	FUNCTION
1	XMINBX	
2	XMAXBX	
3	YMINBX	Virtual coordinate
4	YMAXBX	limits for plot
5	ZMINBX	
6	ZMAXBX	
7	XMINSC	
8	XMAXSC	Screen coordinates
9	YMINSC	limits for plot
10	YMAXSC	
11-25		Unused

RECORD : 7

VARIABLE	NAME	FUNCTION
1	SCLM	Model scale
2	VDIST	Viewing distance
3	TRAN(3,3)	Global-screen coordinate transformation matrix
12-25		Unused

RECORD : 8

VARIABLE	NAME	FUNCTION
1	XMIN	
2	XMAX	
3	YMIN	Coordinate limits
4	YMAX	of model
5	ZMIN	
6	ZMAX	
7-25		Unused

RECORD : 9 UNUSED

RECORD : 10 UNUSED

RECORD : 11

	VARIABLE NAME	FUNCTION
1	NDV	Total number of design variables
2	NINEQC	Total number of inequality constraints
3	NEQC	Total number of equality constraints
4	MAXCYC	Maximum number of design cycles for the problem
5	NCDCYC	Current design cycle number
6	NEVAL	1 .. if function values have to be evaluated 2 .. if gradients have to be evaluated
7	NSTAT	0 .. if in the midst of a design cycle 1 .. if a design cycle has been completed 2 .. if an error has been encountered
8	NACMAX	Maximum number of anticipated constraint violations
9	NAC	Number of violations in current design cycle
10	NACWOB	Number of violations excluding those on bounds of design variables in current design cycle
11	NDVB	Number of blocks of design variables
12	NACB	Number of blocks of active constraints
13	NACWOB	Number of blocks of active constraints excluding those on bounds of design variables
14	NFUNC	Number of function evaluations between restarts
15	NGRAD	Number of gradient evaluations between restarts
16	IBRK	Current breakpoint value
17	NBA	Number of blocks of anticipated constraint violations
18	NFB	Number of blocks of constraint function values
19	NTECH	Optimization solution technique code .. 1 for linearization technique (LINRM) .. 2 for feasible directions technique (CONMIN) .. 3 for gradient projections technique (GRP)
20	DESVAR	Design variable type code .. 1 for member cross section properties as design variables .. 2 for nodal coordinates as design variables
21		Unused
22	IPRINT	Print code
23-25		Unused

****

FILE : job . ELEM

RECORD : TYPICAL
RECORD LENGTH : 25 INTEGERS + 10 SINGLE PRECISION

	VARIABLE NAME	FUNCTION
1	NEU	Element number (user)
2	NES	Element number (system)
3	LTYP	Type of design variable
4	NDV	Number of design variables in element
5	IT	Element type : IT = 1 — ROD IT = 2 — BEAM IT = 3 — TM IT =10 — SPRING BOUNDARY ELEMENT IORD = 1 — SPRING IORD = 2 — TSPRING
6	IORD	Element deflection interpolation order IORD = 1 — LINEAR IORD = 2 — PARABOLIC IORD = 3 — CUBIC
7	IST	Element sub-type IST = 0 — PLANE STRESS IST = 1 — PLANE STRAIN IST = 2 — AXISYMMETRIC
8		Unused
9	NLDREC	Pointer to load record in job.LOAD
10	NCP	Number of corner (attachment) points
11	NGP	Total number of points
12	NSTPT	Number of stress points per layer
13	NLAYR	Number of layers for which stresses exist
14	ISTPTR	Pointer to first record in stress file
15	NMAT	Material type number
16	NTHS	Element property group number
17-22		Unused
23	IREC	Design group number of the first design variable in element
24	IFP	(Forward) pointer to the next element belonging to the same design group
25	ALPHID	Alphanumeric identifier
26	LCP(8)	list of corner points (by system number)

****

FILE : job . MATR

RECORD : TYPICAL
RECORD LENGTH : 5 INTEGERS + 11 DOUBLE PRECISION

	VARIABLE NAME	FUNCTION
1	MATPTR	First record number of material number 'I' (zero if material 'I' nonexistent)

```
2  IMT          Material type :
                 1 - Isotropic material
                <0 - Continuation record (neg. of material type
                     being represented)
3  LRED
4  LGREEN        Color Levels
5  LBLUE
 Remainder of record is dependent on the material type :
 For an isotropic material (IMT = 1 ) :
6  E            Young's modulus
7  VNU          Poisson's ratio
8  G            Shear modulus
9  SY           Yield stress (von-Mises criterion)
10 RHO          Mass density
11              Unused
12 ALPHA        Thermal expansion coefficient
13 U            Thermal conductivity
14 TEMP         Temperature (for temp. dependent material)
15              Unused
16              Unused
**************************************************************************
FILE : job . PROP

RECORD : TYPICAL
RECORD LENGTH : 6 INTEGERS + 15 DOUBLE PRECISION

--------------------------------------------------------------------------
  VARIABLE                  FUNCTION
    NAME
--------------------------------------------------------------------------
1  ITHPTR       First record number of element property group number I
                 (zero if group 'I' non-existent)
2  ITYPE        Element property type :
                 0 - Simple value list
                 1 - Interpolation value list
                 2 - Geometric data is to be interpolated from the
                     following two cross-section definitions stored
                     in this property group
                 3 - Cross-section definition
3  IPTRCS       Pointer to record in this file at which standard element
                cross-section definition is stored (0=none), or at which
                geometric data interpolation data is stored (ITYPE=2).
 Remainder of record is dependent upon the value of 'ITYPE'
   ITYPE = 0
4  LRED
5  LGREEN       Color levels
6  LBLUE
```

7 THS(15) List of element property values; usage depends upon the Element type :
RODS : Cross-sectional area
TM3 : Thickness

****

FILE : job . NODE

RECORD : TYPICAL
RECORD LENGTH : 10 INTEGERS + 14 SINGLE PRECISION + 3 DOUBLE PRECISION

VARIABLE NAME	FUNCTION
1 NU	User number of system node 'I' (negative number flags deleted/merged node)
2 NS	System record number at which to find data for user node 'I'
3	Unused
4	Unused
5 X	
6 Y	Nodal point coordinates
7 Z	
8-13	Unused
14 NDVX	Design variable group number of x-coordinate
15 NDVY	Design variable group number of y-coordinate
16 NDVZ	Design variable group number of z-coordinate
17 NDISC	Displacement constraint code packed into lowest-order six bits.
18-24	Unused
25 NLDREC	Pointer to point load in load file
26 NBL	Number of corresponding unknown block
27 NFR	Relative number of first unknown within block
28 MFP	Map of freedom pattern, packed into the lowest-order twelve bits. From the highest order bit, they are : L(1) - L(6) -- Freedom flags 0 - Suppressed 1 - allowed L(7) - L(12) -- Prescribed displacement flags 0 - Freedom not prescribed 1 - Freedom prescribeD

****

FILE : job . STRS

RECORD : TYPICAL
RECORD LENGTH : 8 DOUBLE PRECISION

This file contains element stress values, one per stress point per layer for each element, and stress resultant values, one per stress point for each element in the model. It is divided into major groups. Each major

group of the file (each loading case) contains one group of records for every stress point in each element. The group contains all stress and stress resultant for all layers of stress values at that stress point. The mode and problem class switches for the model determine which of these sets of data are actually stored in the file. The file contains a total of input "NSTRP" stress records for each loading case.
Each stress point group in this file can contain the following records:

<STRESS RESULTANTS - 1 RECORD>

RST(6) Up to 6 stress resultant components
FREE(2) Unused

<STRESSES - 'NLAYR' RECORDS>

STR(6) Up to 6 stress components
FC(2) Two failure criterions (in percent)
The layers will be ordered from the bottom to the top of the element.

****

FILE : job . LOAD

RECORD : TYPICAL
RECORD LENGTH : 8 DOUBLE PRECISION
Consists of 'NLCT+1' groups of records, each group containing 'NGPT'+1 (logical) records. The first group (Number 0) is reserved for composite loading cases. The second group (number 1) contains loading case 1, etc. The first record in each group is reserved for special purposes, while the remaining records store the load values for each node.

	VARIABLE NAME	FUNCTION
1	VX	Load in X-direction
2	VY	Load in Y-direction
3	VZ	Load in Z-direction
4	MX	Moment about X-axis
5	MY	Moment about Y-axis
6	MZ	Moment about Z-axis
7	RES	Load resultant
8	RESM	Moment resultant

==========

The first record in each group contains:

LDTYPE Type of load group:
0 - Simple loading case
5 - Composite loading case
6 - Composite loading case (at angle 'VAL')
VAL Time or frequency value
MOD Modification flag

****

FILE : job . DEFL

RECORD : TYPICAL
RECORD LENGTH : 8 DOUBLE PRECISION
Identical in structure to load file, but it contains nodal point deflection components and resultants, rather than loads. It also contains a total of 'NLCA' groups of records, rather than 'NLCT' as the LOAD file does. Each record (except the first in each group) contains:

	VARIABLE NAME	FUNCTION
1	U	X-displacement
2	V	Y-displacement
3	W	Z-displacement
4	THX	Rotation about X-axis
5	THY	Rotation about Y-axis
6	THZ	Rotation about Z-axis
7	DIS	Resultant displacement
8	ROT	Resultant rotation

****

FILE : job . SDIR

RECORD : TYPICAL
RECORD LENGTH : 42 INTEGERS
Contains 'NUGRP' records, each describing the submatrix distribution in one row of the stiffness supermatrix.

	VARIABLE NAME	FUNCTION
1	NFR	Number of rows in superrow
2	NFP	Number of first node with freedoms in this superrow
3	NLP	Number of last node with freedoms in this superrow
4	LPD(18)	List showing rows with prescribed displacements: LPD(I) = 0 — Row's freedom not prescribed LPD(I) = 1 — Row's freedom is prescribed
22	NS	Number of submatrix strings in the row
23	IFSPTR	Pointer to the stiffness file record number of first submatrix in row
24	IBAK	Pointer to the earliest preceding row which interacts with the current row
25	LC(2,9)	List of column strings where non-zero submatrices are present. For string 'I': LC(1,I) — Number of first column in string LC(2,I) — Number of last column in string

****

FILE : job . STIF

RECORD : TYPICAL
RECORD LENGTH : 4 INTEGERS + 324 DOUBLE PRECISION

VARIABLE NAME		FUNCTION
1	NR	Row and column position of this submatrix in the
2	NC	supermatrix
3	NFR	Number of rows in submatrix
4	NFC	Number of columns in submatrix
5	SM(18,18)	Submatrix, stored column-wise

****

FILE : job . FUNC

RECORD LENGTH : 1 INTEGER + 18 DOUBLE PRECISION
RECORD : 1

VARIABLE NAME		FUNCTION
1	OBJF	Objective function value
2	EP	Constraint-band thickness
3	VMAX	Maximum constraint violation
4	EPBD	Constraint-band thickness for design bound constraints
5	CONV	Convergence parameter

RECORD : 2	UNUSED
RECORD : 3	UNUSED
RECORD : 4	UNUSED
RECORD : 5	List of integer variable names supplied by the user. Maximum of 36 names, each storing 4 characters.
RECORD : 6	List of double precision variable names supplied by the user. Maximum of 18 names, each storing 4 characters.
RECORD : 7	Values of the 36 integer variables defined in record 5.
RECORD : 8	Values of the 18 double precision variables defined in record 6.
RECORD : 9	UNUSED
RECORD : 10	UNUSED

RECORD : 11 onwards

VARIABLE NAME	FUNCTION
1 NSF	Number of values in subvector.
2 G(18)	Constraint function values.

****

FILE : job . DESV

RECORD : TYPICAL
RECORD LENGTH : 1 INTEGER + 108 DOUBLE PRECISION

VARIABLE NAME	FUNCTION
1 NV	Number of values in the subvector
2 X(18)	Design variable values
3 XL(18)	Lower bounds of design variables
4 XU(18)	Upper bounds of design variables
5 DF(18)	Gradient of objective function
6 DELTA(18)	Direction vector
7 XS(18)	Copy of design variable values during line-search

****

FILE : job . GRAD

RECORD : TYPICAL
RECORD LENGTH : 2 INTEGERS + 324 DOUBLE PRECISION

VARIABLE NAME	FUNCTION
1 NSR	Number of rows in submatrix
2 NSC	Number of columns in submatrix
3 DG(18,18)	Submatrix storing gradients of active constraints with respect to design variables.

****

FILE : job . HIST

RECORD LENGTH : 1 INTEGER + 18 DOUBLE PRECISION
The file is so arranged that the first MAXCYC (maximum number of design cycles) records contain the design history, one record per design cycles, followed by NDVB records (number of design variable blocks) per design cycle containing the values of the design variables at the end of the design cycle.

VARIABLE NAME	FUNCTION

1	OBJF	Objective function value
2	CONV	Convergence parameter value
3	VMAX	Maximum constraint violation
4	STEPSZ	Step size at the end of line-search
5-18		Unused
19	NACTVE	Number of active constraints

The rest of the file contains the design variable values stored as 18 values per record preceded by a header containing the number of values stored in the record.

REFERENCES

Baron, R.J. and Shapiro, L.G., 1980, Data Structures and Their Implementation, Van Nostrand, New York.

Bhatti, M.A., Ciampi, V., Pister, K.S. and Polak, E., 1981, "OPTNSR - An Interactive Software System for Optimal Design of Statically and Dynamically Loaded Structures with Non-linear Response", Report No. UCB/EERC-81/02, University of California, Berkeley.

Codd, E.F., 1970, "A Relational Model of Data for Large Data Banks", CACM, Vol.13, No.6, pp.33-55.

Christiansen, H., 1980, "MOVIE.BYU - A General Purpose Computer Graphics System", Brigham Young University, Provo, Utah.

Date, C.J., 1975, An Introduction to Database Systems, Addison-Wesley, Reading, Mass.

Felippa, C.A., 1979, "Database Management in Scientific Computing - I. General Description", Computers and Structures, Vol. 10, No.1, pp.53-61.

Giles, G.L. and Haftka, R.T., 1978, "SPAR Data Handling Utilities", NASA Technical Memorandum 78701, NASA Scientific and Technical Information Office.

Haug, E.J. and Arora, J.S., 1979, Applied Optimal Design, Wiley-Interscience, New York.

Haug, E.J., Choi, K.K., Hou, J.W. and Yoo, Y.M., 1981, "A Variational Method for the Shape Optimal Design of Elastic Structures", Proceedings of the International Symposium on Optimal Structural Design, University of Arizona, Tucson, Arizona.

Kamel, H.A., McCabe, M.W. and Spector, W.W., 1979, "GIFTS5 Systems Manual", University of Arizona, Tucson.

Kamel, H.A., McCabe, M.W. and DeShazo, P.G., 1979, "Optimum Design of Finite Element Software Subject to Core Restrictions", Computers and Structures, Vol.10, No.1, pp.63-80.

Martin, J., 1975, Computer Data-Base Organization, Prentice-Hall, Inc., New Jersey.

Rozvany, G.I.N., 1980, "Optimality Criteria for Grids, Shells and Arches", Proceedings of the NATO-NSF Advanced Study Institute on Optimization of Distributed Parameter Structures, University of Iowa, Iowa City, Iowa.

Rajan, S.D. and Bhatti, M.A., 1982, "Data Management in FEM-based Optimization Software", Computers and Structures, Vol.16, No.1-4, pp.317-325.

Ross, Daniel, 1966, ICES System Design, The M.I.T. Press, Cambridge, Mass.

Sobieszczanski-Sobieski, Jaroslaw, 1980, "From a Black-Box to a Programming System: Remarks on Implementation and Application of Optimization Methods", Proceedings of a NATO Advanced Study Institute Session on Structural Optimization, Sart-Tilman, Belgium.

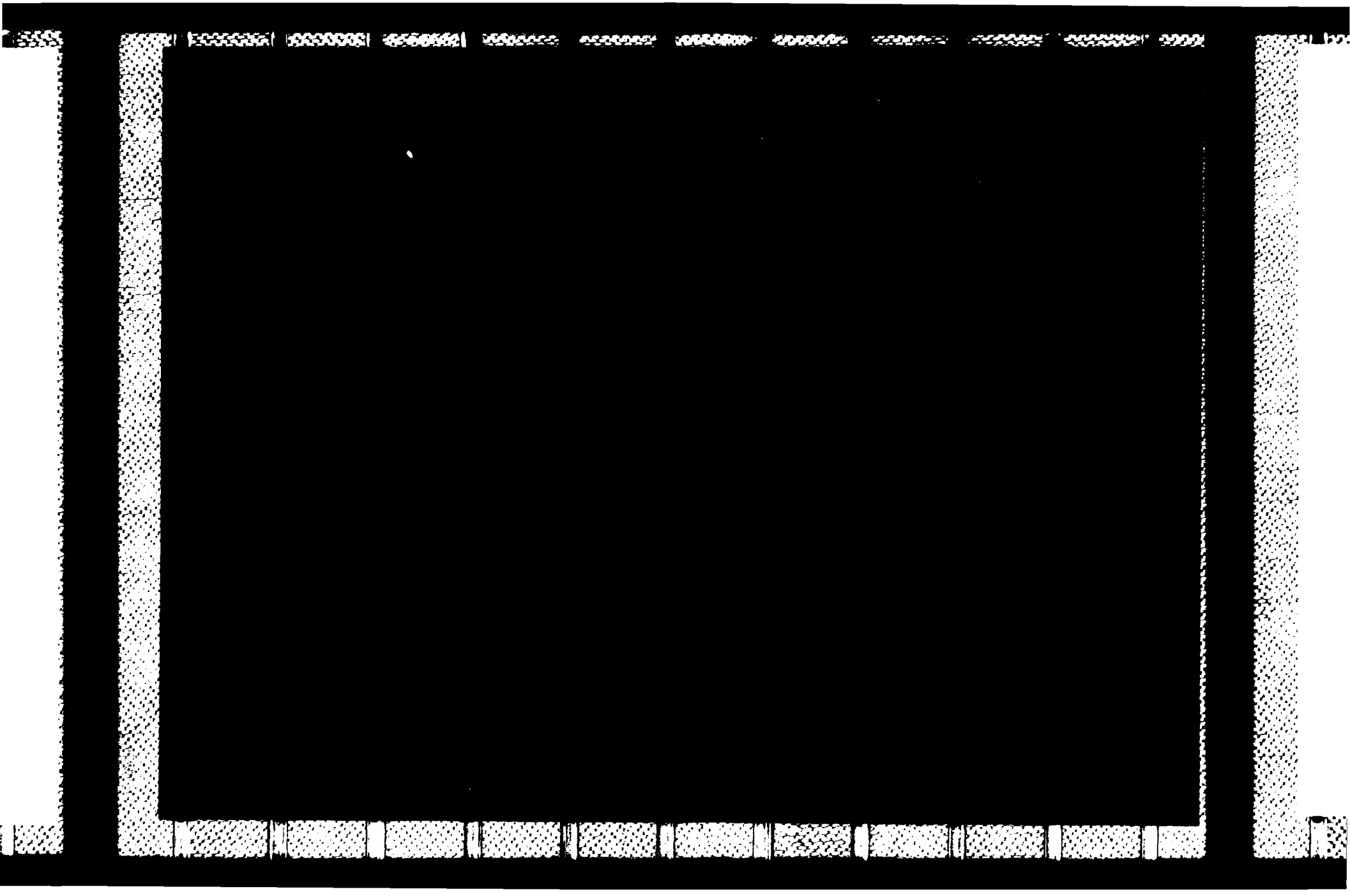
Schrem, E., 1974, "Development and Maintenance of Large Finite Element Systems", Structural Mechanics Computer Programs, Eds. W.Pilkey, K.Saczalski and H.Schaeffer, University Press of Virginia, Charlottesville.

Shaw, A.C., 1974, The Logical Design of Operating Systems,Prentice-Hall,Inc., New Jersey.

Tausworthe, R.C., 1977, Standardized Development of Computer Software, Part 1 Methods, Prentics-Hall,Inc., New Jersey.

Ullman, J.D., 1980, Principles of Database Systems, Computer Science Press, Maryland.

Whetstone, W.D., 1977, "SPAR Structural Analysis System Reference Manual", System Level II, Volume I, NASA-CR-145098-1.



ABSTRACT

This report presents general concepts for database management in design optimization. The need for data management is emphasized. Design optimization procedures use a large amount of data and are iterative in nature. Data used in optimization procedure is described. Logical organization of data using hierarchical, network and relational data models is described with reference to design optimization. Various techniques of physical storage of data are described. A suitable file structure and file operations required for design optimization are given. A comprehensive review of literature for database management in scientific computing is conducted. It is noted that database management ideas are fairly new to engineering community and terminology used in literature varies widely. Well-accepted terminologies are listed. Favorable features and drawbacks of some available database management systems are noted. Based on this study a suitable database management system and a database for design optimization can be developed.

TABLE OF CONTENTS

I. INTRODUCTION

Recent advances in the computer technology have considerably improved the capability of engineer involved in design optimization. Low cost computer systems are available having higher memory, large disk space, interactive capability and graphic display. Such systems were considerably expensive a few years back. Since these new computer systems can be used to handle large quantities of data and perform computation rapidly, it is important to look at the role of data management in design. Many engineering design problems are quite complex. It is impossible for the engineer to specify a suitable design satisfying all the performance requirements and physical constraints without substantial computer analysis. In addition, the amount of information and data used in design optimization computation is so huge that data management becomes extremely important. A database management system can be viewed as both a repository of data used/generated by a given design problem and as a design tool. It offers powerful ways of manipulating data. It liberates the designer from the tedious task of managing data. Thus, a good database system in computer-aided design provides a tool for the designer in achieving results in an efficient and systematic way.

Computerization in commercial field in the areas such as business accounting, inventory control, and task scheduling has been quite successful with application of powerful data management systems. However, due to the complex nature of information in scientific applications, the growth of data management system has not taken place

to the extent as in business applications. It appears that the analysis capability in computer-aided design of structural and mechanical systems is enhanced with the use of some database management systems that are now available (Ulfsby, et al., 1979; Fischer, 1979; Comfort, et al., 1978). These database management systems are highly specialized in nature. Extensive modification of these systems are required to use them in design optimization.

Current trends in database management are toward arriving at a database management system that meets certain standard requirements, such as data independence, flexibility, mulitiple usage, and non-redundancy. These requirements can only be met by proper design of a database. Different methodologies having markedly different characteristics and features are available for database design (Buchmann and Dale, 1979). However, most of these methodologies have their origin in business environment. A database that suits design environment can be obtained using a suitable scheme. Based on a detailed study of the requirements, and data used in design optimization, appropriate database management concepts can be developed. This forms the subject matter of the present report.

In Chapter II the need for database management system in design is given. Data used in design optimization is discussed in Chapter III. Structural and mechanical system design is used as a model. Data similar to these systems are needed in other applications. In Chapter IV, different data models used in data representation are discussed. Also, a data model suitable for design optimization is given there. File structures that are commonly used in database management field and

addressing techniques are given in Chapter V. A comprehensive review of literature on engineering database management is given in Chapter VI. Finally, discussions on database management concepts are given in Chapter VIII. References cited in the report are given at the end of the report. There is one appendix to the report. It contains various terminologies used in hardware, and logical and physical database structures.

II. NEED FOR DATABASE IN DESIGN OPTIMIZATION

Computer programs for optimal design of large structural and mechanical systems can be developed for automatic computation based on well known design optimization methods (Haug and Arora, 1979). Finite element techniques are usually adopted to analyze the system within a design iteration. As such the finite element technique require huge amount of computation and data storage depending on the size of problem at hand. Further, the amount of data handled depends directly on the number of iterations performed in iterative design optimization algorithms. Therefore, a careful consideration of data handling aspect is necessary in design optimization.

It is important to realize that engineering design optimization and engineering analysis are fundamentally different in nature. In analysis, it is generally assumed that a solution exists and numerical methods used are stable. Also, many engineering problems require the use of data only a few times during the solution procedure. In optimal design on the otherhand we find solutions in an iterative manner. Existence of even a nominal design satisfying constraints is not assured, much less existence of an optimal design. Therefore, it becomes essential for the designer to exercise control over the suitable design optimization method that has to be used. In such a case, the data used by one method should be made available for use in another method. The concept of centralized database becomes important. A centralized database which allows interaction between a finite element program and an optimization program can be used to improve design

iteratively. Such a database provides an option for the designer to interrupt the program execution and provides flexibility for the designer to change his design parameters. A properly designed database when used with interactive computer graphics, offers considerable aid to the engineer involved in design optimization (Galletti and Giannotti, 1979; Somekh and Kirsch, 1979).

III. DATA USED IN DESIGN OPTIMIZATION

In optimal design of structural and mechanical systems, we generally use nonlinear programming techniques (Haug and Arora, 1979). The design objectives and constraints for these systems are specified quantitatively and expressed in terms of a mathematical model. Design of a system is specified using a set of parameters called design variables. The design variables depend on the type of optimization problem. In design of aircraft components such as stiffened panels and cylinders, the design variables are spacing of the stiffeners, size and shape of stiffeners, and thickness of skin. In optimization of structural systems such as frames and trusses of fixed configuration the sizes of the elements are design variables. Thickness of plates, cross-sectional areas of bars, moment of inertia represent sizes of the elements. If shape optimization is the objective, the design variables may include parameters related to geometry of the system.

The optimization problem deals with minimization or maximization of objective functions subjected to certain constraint conditions. The constraints may be classified into performance constraints and size constraints. The performance constraints are stresses, displacements, local and overall stability requirements in static case and frequencies and displacements in dynamic case, flutter velocity and divergence in aeroelastic case or combination of these. The size constraints are minimum and maximum limits on design variables. In nonlinear programming, the search for the optimum design variable vector involves iterative scheme. The design variables data at n and n+1 iterates are

computed. The direction of travel vector and step size are computed. The direction of travel involves computation of gradients of objective and constraint functions with respect to the design variables. Data belonging to equivalent design variables are grouped there by reducing the size of design variable vector.

In most problems of structural and mechanical system design, behavior of the system can be defined using state variables, e.g., stress and deflection. In such a case, state space formulation is frequently employed (Haug and Arora, 1979). Design sensitivity coefficients in terms of matrix equation are determined in state space formulation. Adjoint equations are used to define a set of variables that provide design sensitivity information. Symmetric matrix equations can be used to advantage thereby reducing the data storage requirements.

In parametric optimal design problem yet another set of variables called the environmental parameters are used (Haug and Arora, 1979). Optimal design problem is formulated with additional constraints called parametric constraints. The solution of parametric optimal design is obtained in two steps (1) the solution of subproblem and (2) solution of outer problem. The data of these problems may be stored separately.

Finite element method and numerical methods are adopted during analysis of structural and mechanical systems. Finite element method uses data such as element number, nodal connectivity, element stiffness matrix, element mass matrix, element load matrix, assembled stiffness, mass, and load matrices, displacement vectors, eigenvalues, eigenvectors, buckling modes, decomposed stiffness matrix, and the stress matrix. In general data used in finite element is quite large.

Symmetry of stiffness and mass matrices is taken into account so that data storage requirement is reduced. Hypermatix schemes are generally used in dealing with large matrix equations.

For design of large structures, efficient design sensitivity analysis is particularly critical. For such structures, substructuring concept can be effectivly integrated into structural analysis, design sensitivity analysis, and optimal design procedures. In this concept, one deals with small order matrices as the data can be organized substructure-wise. The degrees of freedom can be classified into boundary degrees of freedom and interior degrees of freedom. Data for the stiffness matrices corresponding to these degrees of freedom can be separately stored. Data of constraint functions corresponding to internal and boundary degrees of freedom are used in determining design sensitivty calculations. Adjoint matrix data is stored for each substructure.

In case of multiple loading conditions, performance of the system is determined for each loading condition. Design sensitivty data is computed for each violated constraint. To reduce the size of data, the constraints which are not critical at optimum point are deleted. In case of fail-safe optimal design problem, data of state-equations and constraints are generated for each damage condition. Optimal design of a system under dynamic loads requires additional data of eigenvalues and eigenvectors. The eigenmodes data are used in modal anlysis to reduce the size of equations in design sensitivity analysis.

Many real world problems will have features that are not explicitly contained in general optimal design formulation. Problems with peculiar

features need to be treated by making minor alterations in the general algorithm. Interactive computation and graphics can be profitably employed in design optimization. At a particular iteration, the designer can study the data of design variables, constraints which are active, performance of the system, cost functions, admissible direction of travel, sensitivity coefficients, etc. The designer can make judgement regarding suitability of a particular algorithm, necessary change of system parameters, redefinition of convergence parameters and use them in achieving the optimal design. Interactive graphics display facilities require additional data for display of system model, results, and graphs.

Thus, for design optimization, data generated during analysis must be saved. The data saved is used for formulation of constraints. Constraints are checked for violation. Design sensitivity analysis of violated constraints is carried out. Calculation of design sensitivity coefficients needs most of the data generated during analysis. Therefore, data must be organized and saved properly in a database for efficient design optimization.

IV. DATA BASE ORGANIZATION FOR DESIGN OPTIMIZATION

Usually in any design environment, the design of several engineering systems is carried out simulataneously. Also, in case of large engineering systems, a number of groups are involved in design of various subsystems. It is desirable to create a separate database for each subsystem or project. The subsystem database can be identified using a name. Each project may consist of a number of tasks. Depending on the task complexity and volume of data generated, database may be subdivided into a number of data libraries. These data libraries in turn store a number of data sets that are identified by a name. The individual groups dealing with the subsystem design are authorized to access, store and modify data in a particular database, thereby ensuring database security.

Modular program organization is essential for efficient design optimization algorithm. A module is a software element that performs a well defined task. These modules must possess exact knowledge of the data structure. It is essential to study various data structure organizations and choose those data structures that are most suitable for design optimization.

4.1 DATA MODELS

The data models are logical representations of the data utilized by the users of the database system. Data should be represented in a form that is most convenient from users point of view. A data model stores the data as well as relationship between the data items. The overall logical database description is referred to as a schema (overall model

of data or conceptual model). A schema is a chart of the types of data that are used. It gives the names of the entities and attributes and specifies the relation between them. There are three types of data models that are commonly used. They are (1) Hierarchical model, (2) Network model, and (3) Relational model. These three types of data models are frequently employed in business application. Suitability of these models with respect to design optimization is considered in the following paragraphs.

4.1.1 Hierarchical Model:

In this model the data is represented by a simple tree-structure. A tree is composed of hierarchy of elements called node. Every node has one node related to it at a higher level. The node at a higher level is called a parent node. Each node can have one or more nodes related to it at a lower level called children. A node at the top of a tree is called the root. Figure 4.1.1 shows a hierarchical model. An elementary hierarchical relation is the one in which there exists one and only one parent for a set of child nodes. The root of an elementary relation has no parent. A hierarchical model is a collection of elementary hierarchical relations. Hierarchical model has one-to-many relationships. Hierarchies is a natural way to model truly hierarchic structure in real world problems. Tree structures are used both in logical and physical data descriptions. In logical data descriptions they are used to describe relations between record types. In physical data

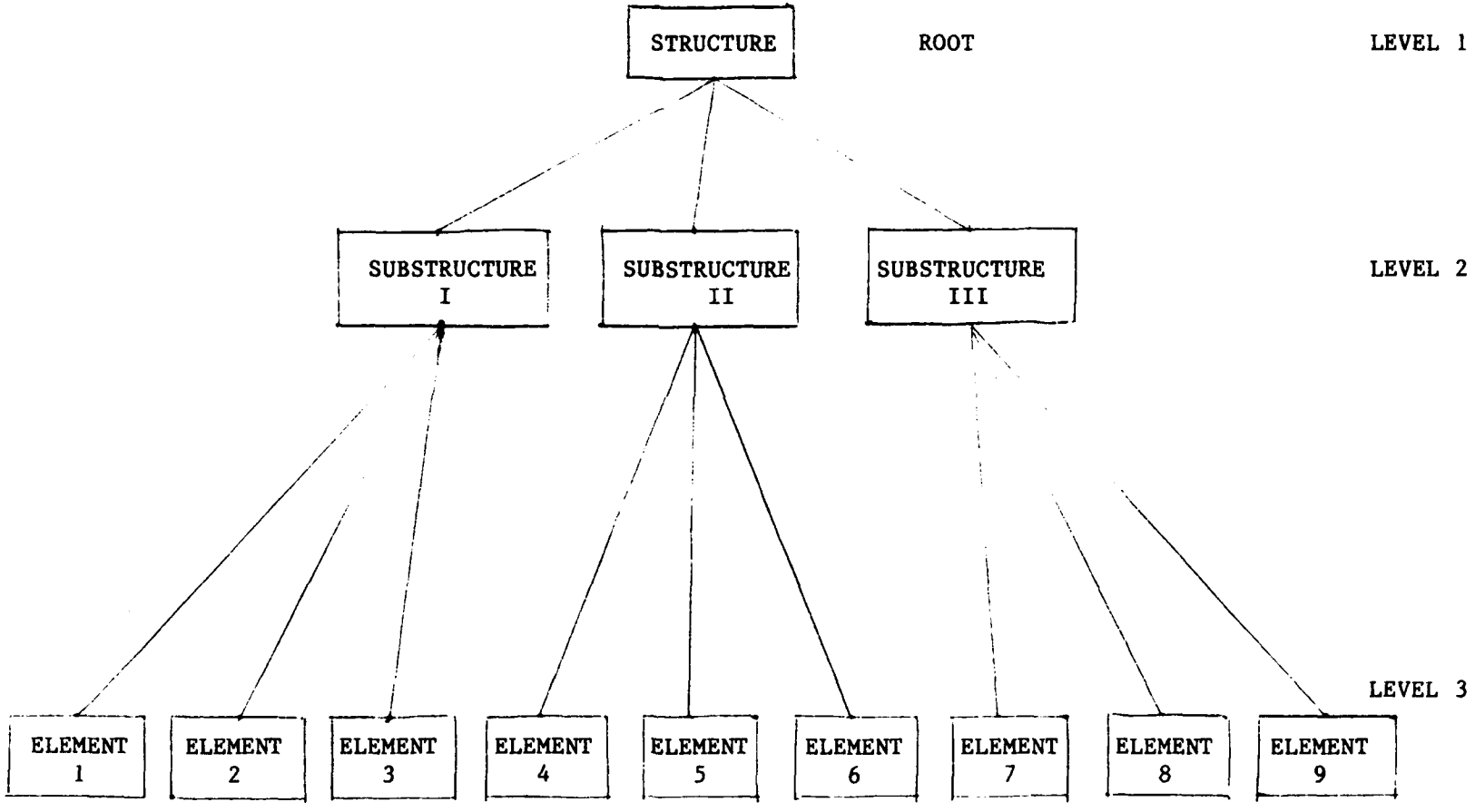


Figure 4.1.1. A Hierarchical Model.

description they are used to describe sets of pointers and relationship between entries in indices. A hierarchical file has a tree-structure relationship. Hierarchical structures are relatively easy to represent in many applications. In many other applications, data are not tree-structured. Therefore, in such cases, other complex models have to be used to represent data. The hierarchical model simplifies the software although the corresponding system is sligthly lengthy.

In finite element analysis, we can form a hierarchical model with data items such as structure, substructure, elements and nodes (Figure 4.1.1). Another example of hierarchical model is in representation of structural stiffness matrix. Depending on the size of matrices two or three levels of data can be used. The hierarchy is established by the two levels of data which actually contain the matrices.

4.1.2 Network Model:

A collection of arbitrarily connected logical relations is called network relation. The data model defined by such a network is called the network model. A network is more general than a hierarchy because a node may have any number of immmediate higher level relationships. Any data item in a network structure can be related to any other data item. Figure 4.1.2 shows a network model. The network model can have one to many levels of data representation as with hierarchical models. It is possible to make a hierarchical data model non-hierarchical by adding new segment types and new directional logical relations. Network allows many-to-many relationships.

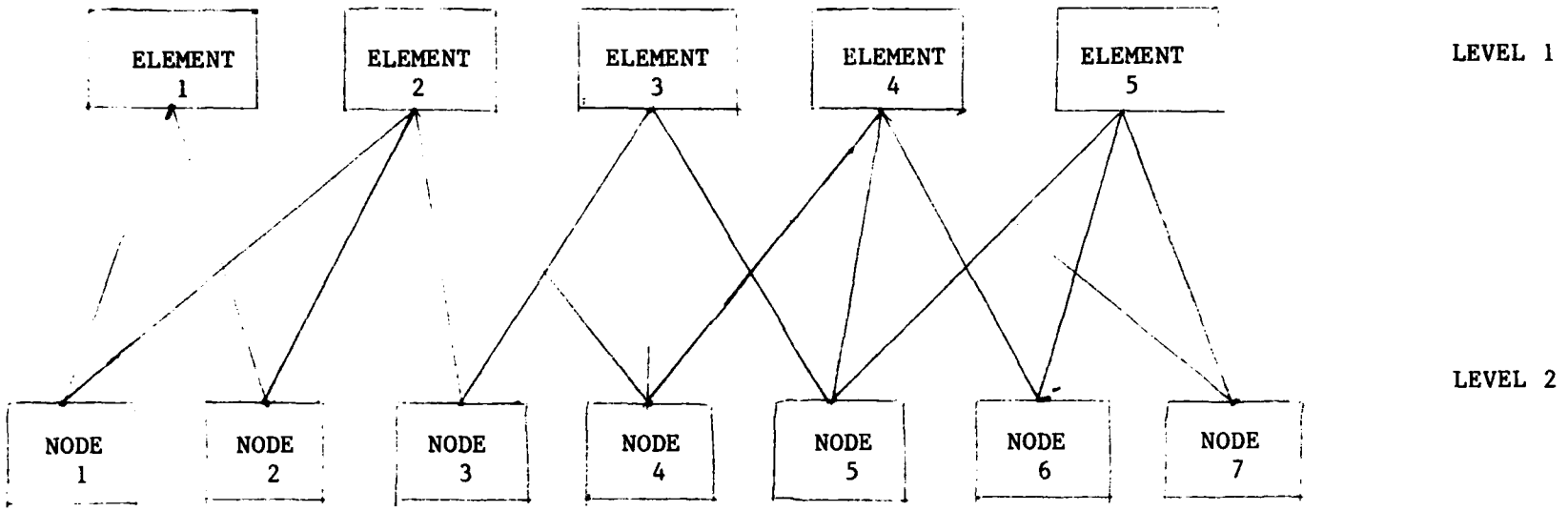


Figure 4.1.2. A Network Model.

A network relation is said to be simple if each directed logical relation is functional in at least one direction. This means a schema in which no line has double arrows in both direction. Complex network relation will have double arrows going in both directions. For example in finite element analysis we can form data model with items such as elements and nodes. The network data model allows complex relationships that commonly occur in real world problems. The disadvantage of this model is in its complexity and in the associated data description language.

4.1.3 Relational Model:

Tables are the most convenient way of representing some data. If a set of attribute pairs are properties of an entity occurence, then they are said to be logically related. A data model constructed using relations is referred to as a relational data model. A relational data model is constructed from a tabular representation of data. Figure 4.1.3 represents a relational model of data. The rows of the table are generally referred to as tuples. The columns are referred to as attributes. If there is one tuple, the relation is said to be unary. A binary relation has two tuples. Relations of degree n are called n-ary. The relational model provides an easy way to represent data. This model can be easily implemented as physical storage of tabular records are less complex than hierarchial and network models. The database can be expanded easily with additions of tuples and attributes. The operations such as PROJECT, JOIN, and SELECT can be used to form new relations. Examples of relational models in finite element analysis are

TRIG

ELEMENT NO.	MATERIAL NO.	NODE 1	NODE 2	NODE 3	THICKNESS

NODE

NODE NO.	DOF 1	DOF 2	DOF 3	DOF 4

CORD

NODE NO.	X	Y	Z

Figure 4.1.3. A Relational Model.

element-node relation, and node-degree of freedom relation. TRIG relation in Figure 4.1.3 contains tuples of triangular element data. There are six attributes of this relation namely element numbers, node numbers, material number, and thickness. The relation NODE and relation CORD contain data of node numbers and coordinates of nodes, respectively.

Easy access to data in relational model is an important feature that is not available in other model. In a relational model high degree of data independence can be achieved. Also, in a relational database it is simpler to develop and implement data query and data manipulation languages.

4.2 DATA MODEL FOR DESIGN OPTIMIZATION

The data used in design optimization was described in Chapter III. It can be seen that majority of the data is numeric. It can be classified into tabular and matrix forms. In general data can be grouped into scalars, arrays, tables, matrices and character strings. These data can be grouped together to form a data set. A data set is a collection of data items. Data items in a data set are ordered by the users or application programmers in a way that is convenient to use. The user or the application programmer can describe his view of data in terms of relations between data items in a data set. Data sets are named and indexed to identify them in the database.

In design optimization, the concept of design variables is fundamental and all data structures should be built around it. A hierarchical data model for representing the design variables is

described as follows: Figure 4.2(a) shows different levels in a hierarchical model. The first level in the model is called DESVAR. It provides basic information about the design variable and its location in the structure. Attributes of the system are described in rows of the table. The rows in the table indicate the substructure number, equivalent substructure number, design cycle, and type of optimization problem. Columns represent the values of the attributes for a specific substructure. Since the values of the design variables cannot be represented by a single value, a pointer is used to show the location where the design variables are located. The second level of the data model represents the design variable group data DESGRP. The first row represents the design group linking, the second row represents the material number, and the third row indicates an address to the lower level. The columns represent design group numbers. The third level in the data model is DESNUM. The rows in the DESNUM are the design variable numbers and design variable cross-section type. The fourth level in the hierarchical model is DESDAT. It contains the actual data for design variables, i.e., cross-sectional data for the example.

A use of the relational data model is described with an example of the finite element analysis. For the finite element idealization, structural element numbers, material number, and nodal connectivity are stored in the relation named ELMT. The nodal coordinates and degrees of freedom numbers are stored in the relation NODE. These relations are shown in Figure 4.2(b). The relation ELMT can be manipulated to form new relations MAT and ELNOD using the PROJECT operation of relational

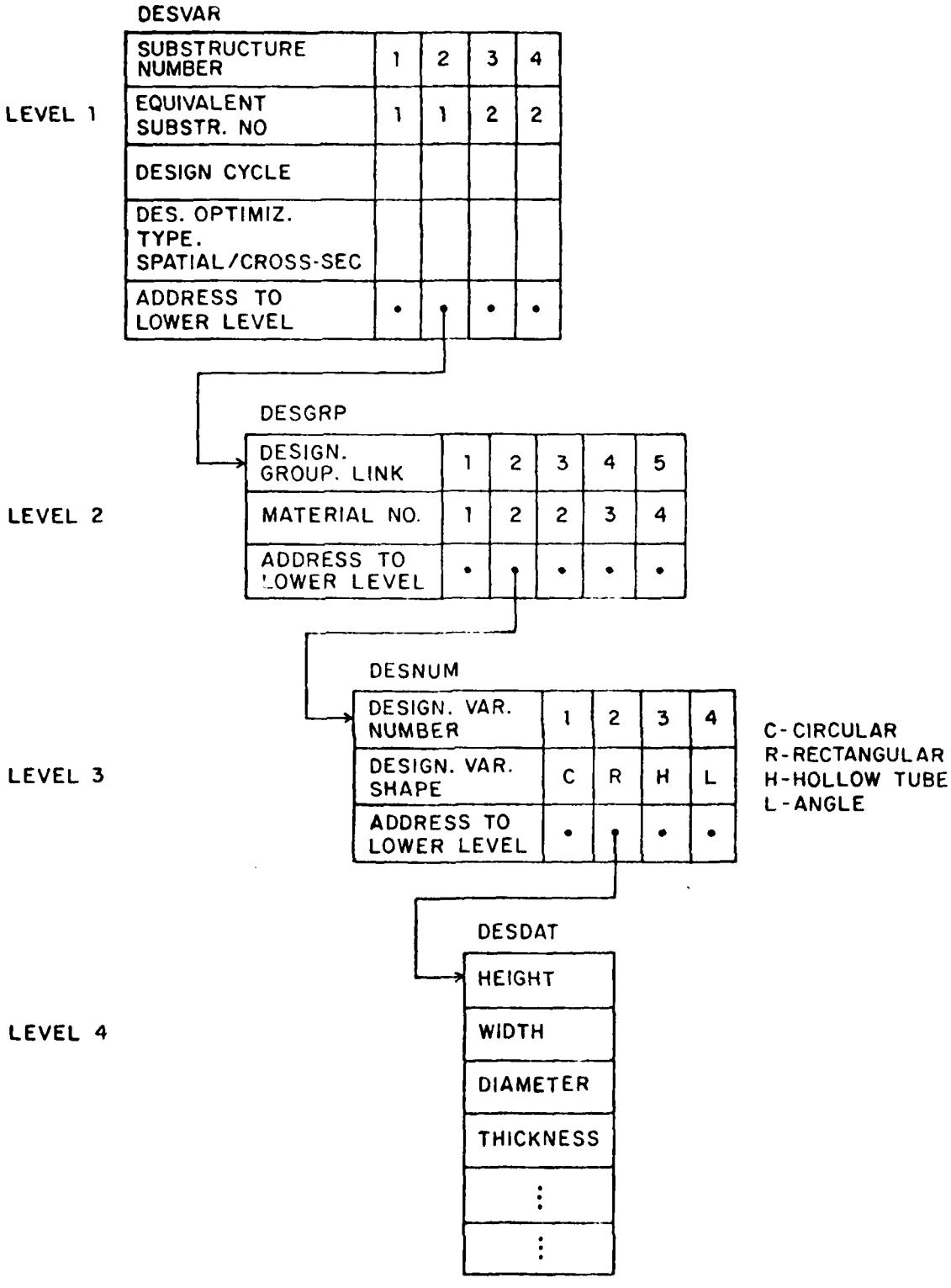


Figure 4.2(a) A Hierarchical Model for Design Optimization

algebra. NODE relation can be projected into DOF and CORD relations. Using JOIN operation, the relations ELNOD and DOF can be combined to form a new relation ELDOF. The application programmer can employ different relations to those defined in the schema. Thus relational model provides more flexibility to the database user. However, indiscriminate use of PROJECT and JOIN operations can produce invalid results in some cases.

Another example of the hierarchical data model can be given for the hypermatrix data. In storage of large order matrices encountered in many finite element applications, it is convenient to subdivide matrices into smaller matrices. These small sized matrices are called sub-matrices. It is possible to organize submatrices into various hierarchical levels. Figure 4.2(c) represents a hierarchical scheme of data model representing a hypermatrix. This scheme of representing hypermatrix helps in solving large order equations in a small memory environment.

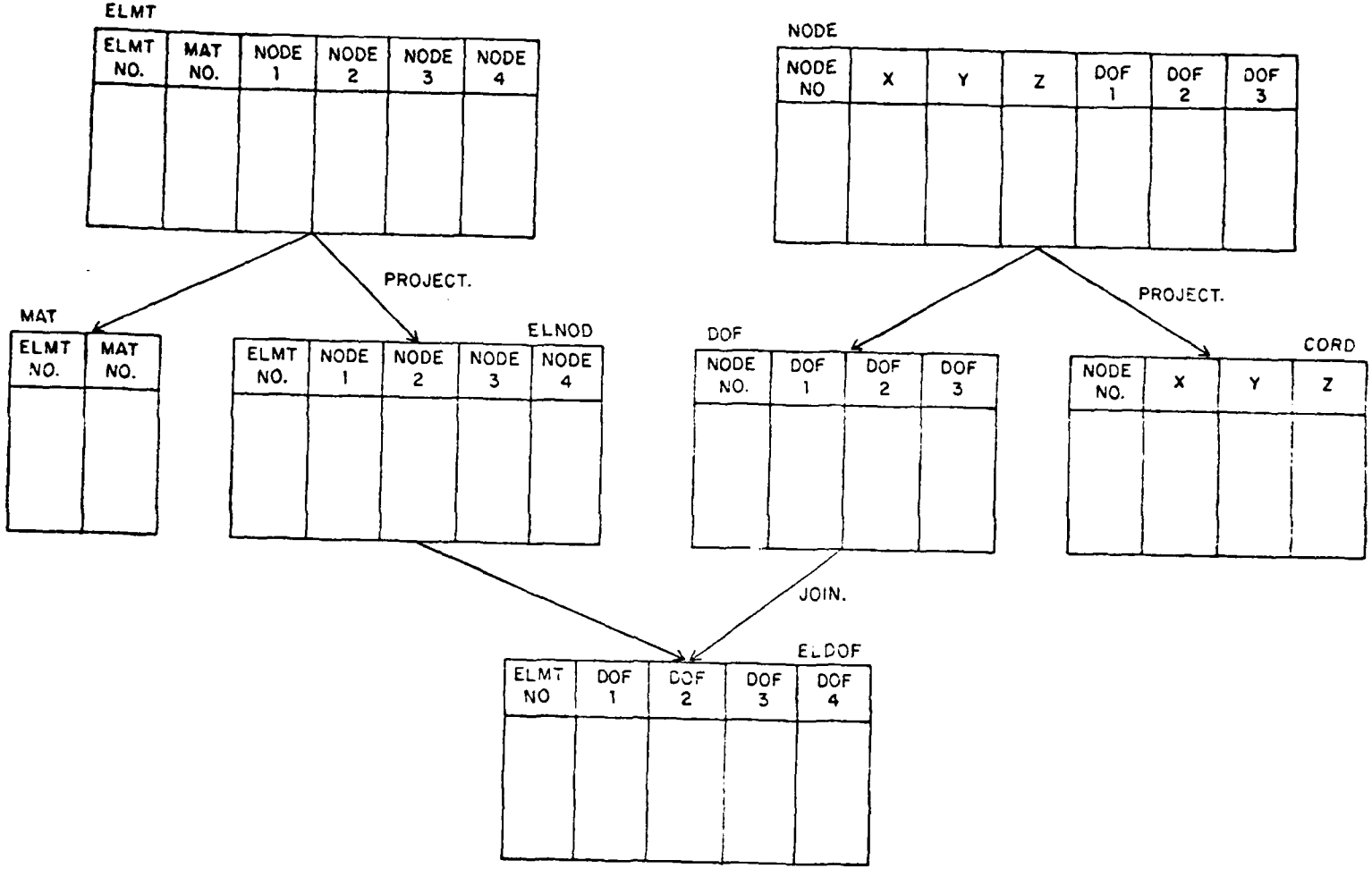


Figure 4.2(b) Relational Data Model for Finite Element Analysis

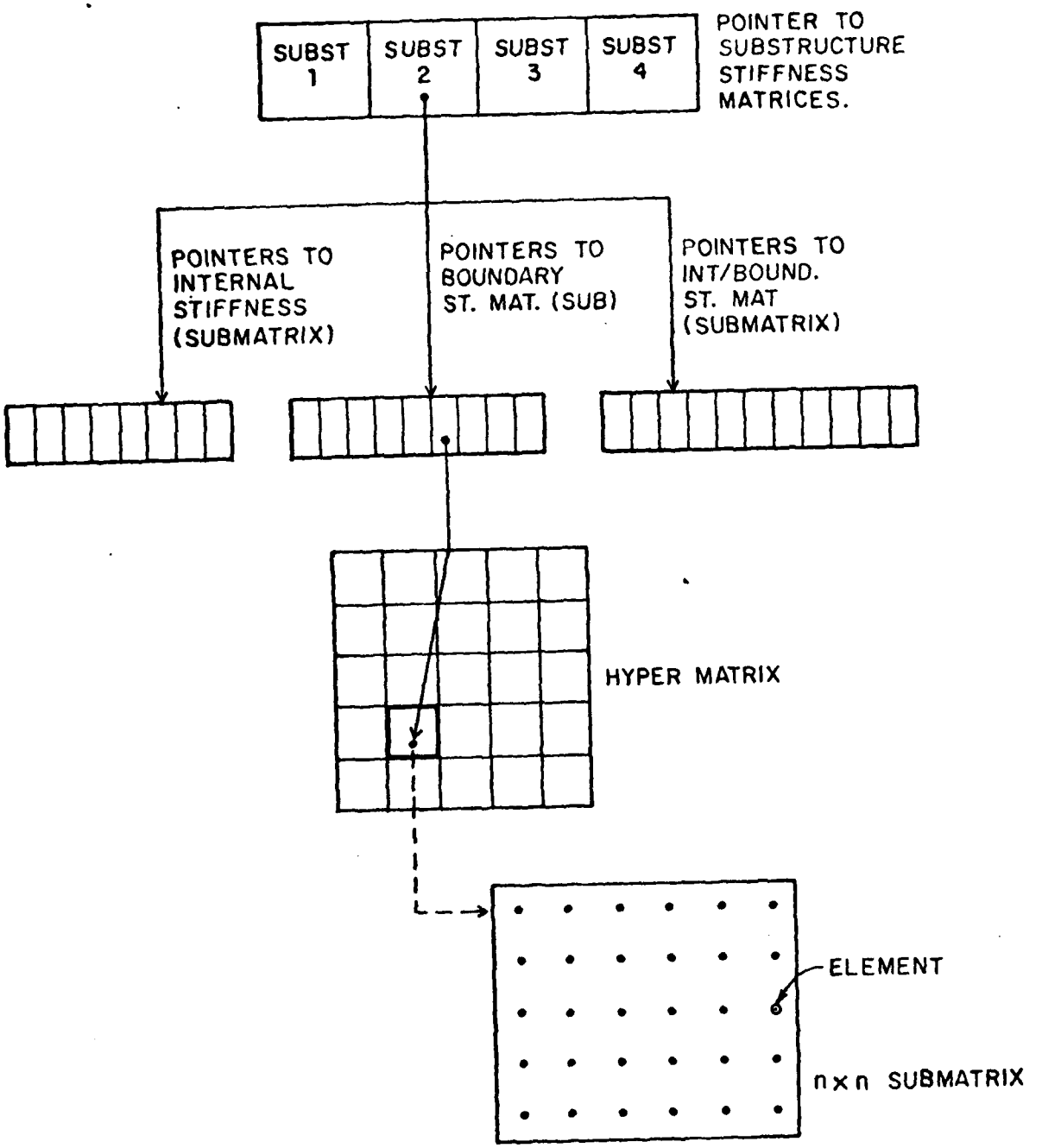


Figure 4.2(c) Hierarchical Model for Hypermatrix Representation

V. FILE STRUCTURE

In this chapter, different types of file structures currently used to store data on an auxillary storage device are reviewed. Auxillary storage means tape drives, disc drives, and disc drums. Tape drives are termed sequential device, whereas disc drives and drums are called direct access devices. The physical storage of data on these devices are generally different from the logical form of the data. Various aspects of storage techniques namely operational efficiency, response time and cost have to be studied before a particular technique is adopted. Various file-storage techniques are presented in the following paragraphs. Also, storage of data on physical devices based on concepts such as 'blocking', 'record length', 'paging', 'addressing', and 'pointers' are discussed.

5.1 SEQUENTIAL FILE

Sequential file is usually associated with magnetic tape. However it can also be used on a direct access storage device. A sequential file is a collection of records which are stored one after another in a sequential manner. Generally, a sequential file (magnetic tape) is used to hold large quantities of data. Also, sequential storage devices are less expensive. Limitation with sequential file organization is that access time for an individual record is dependent on the location of record in the file.

5.2 DIRECT ACCESS FILE

A direct access (random access) file is usually associated with disk or drum. It is a collection of records in which the time required to locate a particular record is independent of that record's position in a file. Records in a direct access file can be retrieved or stored by specifying the actual address of records. There are several ways of accessing a record. One method is to specify the relative record number (from the start of a track and record number respectively) of a desired record. A second method is by specifying relative track and a record key. Another way is by specifying the actual cylinder head number, and track record number on the device. Direct access device can be used to store many kinds of file organization for example sequential, direct, and indexed sequential. Records in a direct access file can be retrieved, modified and stored in any desired order. This capability provides rapid transmission of data between main memory and direct access device. A primary advantage of a direct access file is the speed with which an individual record can be accessed (or stored) in a large file. This advantage must be weighed against cost of direct access device.

5.3 BLOCKING

A group of data which forms a physical record is referred to as a block. Logical records are usually grouped together into blocks and are stored and retrieved one block at a time. Blocks are later split into actual logical record in the program working area. This helps in increasing efficiency in storage space and time involved for storage and retrival operations.

5.4 RECORD LENGTH

Fixed length records are desirable as they are simple to use and less complicated to program. There are situations in which logical records can be of variable length. For example records of element numbers with nodal connectivity numbers; records of node numbers with degree of freedom numbers for each node. In such situations, the physical records will be filled with as many logical records as possible. The gap leftout in the physical record has to be minimized by adjusting the physical record length.

Sometimes, it is required to store a variable length of list of values of the same attribute. For exammple, number of nodes connected to an element varies depending on the type of element used. In such cases, it may be advantageous to store the data as a string of values to reduce the storage space. The program logic will be simplified if the maximum length of list is used to store variable length attribute list.

5.5 PAGING

Blocks of data known as a page is transferred between the main memory and the peripheral storage device. The concept of a page helps in minimizing the seek time. Page must be used in a good database management systems.

5.6 POINTERS

Links betweeen one record and another can be established with pointers. Pointers are stored in the record itself which indicate where another record is located on the physical storage. Three different types of pointers - machine address, relative address, and record

identifier - can be used. Records in a file may be sequentially numbered. The pointers are simply the sequential number. Sequential number may later be converted into machine address. Machine independence can be achieved to a certain extent through the use of relative address pointers. However, machine address pointer is faster than the other two types of pointers. In record identifier method, the address must be established using symbolic pointers. However, this method is slower, especially if the record has an addressing method needing more than one seek.

5.7 ADDRESSING TECHNIQUE

Records in a file can be identified and located using a unique number or a group of characters called the key. There are several methods of addressing a record. The simplest way of locating a record is to scan the file and locate the key of each record. This method is too slow. Another method is the 'Block Search' technique. In this technique a defined number of blocks of records are skipped and then searched for the block containing the required record. Files are addressed by means of a table called an index - the indexed sequential file method (Fig. 5.7). The input to the table is the key of the required record and result of the index table search is the address of the required record. By this method, considerable time is saved but space is needed to store the index. Hashing is another form of addressing technique. In this method, item's key is converted into near-random number and the number is used to determine where the record is located. There are other addressing techniques such as indexed

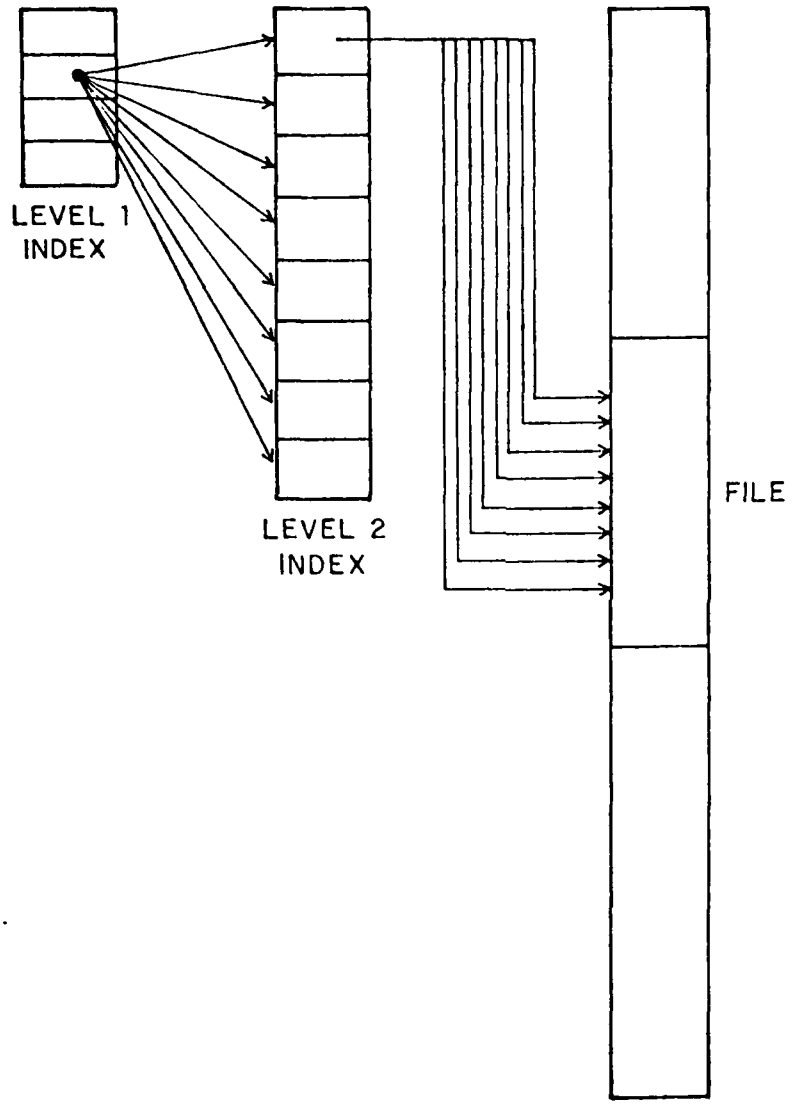


Figure 5.7 Indexed Sequential Addressing

nonsequential technique, key-address technique, and algorithm technique. However, any of the combination of techniques can be used for addressing purposes.

5.8 FILE STRUCTURE FOR DESIGN OPTIMIZATION

Based on the file storage techniques as defined in the above paragraph, a file structure for design optimization can be developed. Data can be stored, retrieved and modified using certain standard operations/functions such as define a file, open a file, define a data set name, write a data set, read a data set, and check for existence of a file. The status structure of a file can be defined; for example FILE: existing/non-existing, opened/closed, RECORD empty/written. The constraints for calling various functions must be defined: (1) file must have been defined before it can be opened or written, (2) file must have been opened and record must have been written before it can be read or modified, (3) a record once written cannot be emptied again, and (4) a closed file can be opened again or deleted. A status diagram for a file system is shown in Figure 5.8.

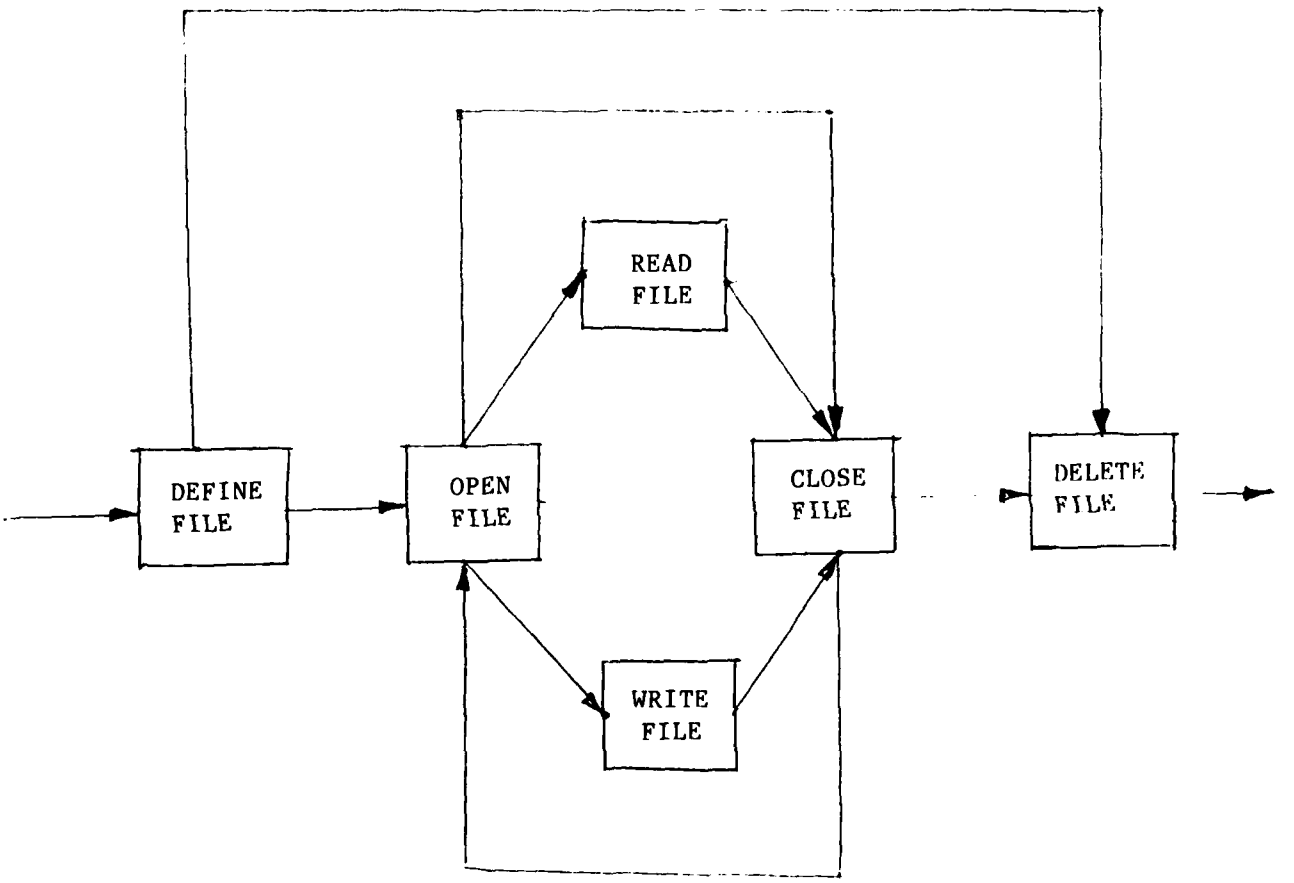


Figure 5.8 A Simple File System.

Various operations on a file are:

DFDEFN - Define a file in user's directory

DFOPEN - Open a file in the directory

DFDELE - Deletes a file

DFCOMP - Compress a file

DFINFO - Give information of a file

DFCLOS - Closes a file

DSDEFN - Define a data set in a file

DSRDFN - Redefine a data set order and size

DSDELE - Delete a data set

DSCOPY - Copy a data set to another data set

DSGET - Get a data set

DSPUT - Put data into data set of a file

Some basic files required for design optimization are given as follows:

INTL - Cost function, constraints, optimization parameters

DESV - Design variable file

ELMT - Element numbers and nodel connectivity

NODE - Node numbers and degrees of freedom numbers

STIF - Stiffness file (assembled stiffness, geometric stiffness, etc.,)

MASS - Mass file (assembled lumped, consistent mass, etc.,)

LOAD - Load file (acceleration, concentrated, distributed, etc.,)

MATR - Material file

DEST - Decomposed stiffness

SOLT - Solution for displacement, stress, frequencies, mode shapes etc.,)

DSEN - Design sensitivity file (adjoint matrices, gradients etc.,)

OPTZ - Optimization file (optimal solution, history of cost function, constraints, design variables, violation of constraints etc.,.)

If substructuring technique is adopted, then some of these files could be organized separately for each substructure.

VI. REVIEW OF LITERATURE ON DATABASE MANAGEMENT IN ENGINEERING

A number of papers and reports have been published on database management in recent years. It is important to take advantage of what has been done in the field of database management. Therefore, a comprehensive review of literature on the subject is presented in this chapter. The review is limited to the database management application in scientific and engineering computing. The review may be broadly grouped into two categories: (1) database management concepts, methodology, etc., and (2) database management system that are currently available.

6.1 DATABASE MANAGEMENT CONCEPTS

The database management concepts are not new in the area of business computing. However, the database management concept is particularly new in the area of scientific and engineering community. During last five years a number of papers have been published on the database management in scientific computing. The paper by Fellipa (1979) on database concepts in scientific computing highlights the difference between the business data management and scientific data management. The importance of centralized database is stressed in the paper. The terminology used in the business computing is fairly new in scientific computing. A comprehensive list of terminology that is relevant to scientific computing is given in another paper by Fellipa (1980).

An interesting paper by Bell (1982) gives some introduction to data modelling in scientific computing. It discusses some issues on how

scientists actually use their data. The paper also gives some comparison between data modelling for scientific application and commercial application.

Several research studies have been conducted to find out a suitable way to design a database management system for scientific and engineering applications. Buchmann and Dale (1979) analyzed different methodology for database design. Various steps involved in a database design are brought out. Also, they have presented a framework for evaluating database design methodology. Roussopoulus (1979), designed CSDL - an interactive language for designing conceptual schema of databases. A database system architecture for interactive computer-aided design has been proposed by Kunii (1979). Grabowski and Eigner (1979) in their paper have discussed various issues in designing a semantic data model. Difficulties of information modelling in CAD applications is brought out in that paper. Another paper by the Grabowski, Eigner and Rausch (1978) on CAD data-structure for minicomputers brings out various steps involved in database design process.

Lafue (1978) discusses several problems related to design of a database system in his paper. He addresses several issues related to semantic, integrity, consistency, maintenance, and manipulation of a database. In another paper by the author (1979), merging of data definition language and data manuipulation language has been suggested. The reason mentioned is that the database schema is continously redefined, i.e., record types added, deleted and modified. Therefore, it is better to combine the two languages.

Proceedings of a NASA conference (Publication 2055, 1978) on Engineering and scientific data managemnt provide a wealth of information on the subject. A good review of requirements for a database managemnt system in support of scientific and technical application is made by Lopatka and Johnson, (1978). In particular, the paper deals with the data management system for CAD/CAM needs. The proceedings also include some papers on database design methodology and applications. A detailed discussion on conceptual design of a data management system is presented by Elliot, Kunii and Browne (1978). The paper describes a system design based on hierarchical model of data. The data definition language used in the system is described in detail. The paper also gives some practical examples on structural design and wind tunnel data management. Application of data management system for weight control was presented by Bryant (1978). Management of atmospheric data was presented by Jenne and Joseph (1978).

A fundamental coverage on data structure, data definitions is presented by Browne (1976) in the third ICASE conference on scientific computing. The paper includes details of storage mapping functions, and data representation with some examples. A storage mapping function that utilizes an inverted file system is also given in the paper. Data management concepts applicable to aircraft industry and its CAD/CAM requirements are explained by Fulton and Voight (1976).

Studies on data structures include mainly hierarchical and relational data structures. Lopez (1974) developed a hierarchical database management system called FILES. This DBMS is primarily for applications in structural engineering. Jumarie (1982) has shown that a

hierarchical data model simplifies the software in development of a decentralized database with the use of microcomputers. Fishwick and Blackburn (1982) dicussed some advantages of a relational data model from an engineering point of view. The paper reports the use of a relational data management system in an integrated design system called PRIDE. Blackburn, Storaasli and Fulton (1982) in another paper further demonstrate the use of relational database management system in CAD. Some engineering test problems using relational data management system are presented in the paper. Haskin and Lorie (1982) of IBM Research Laboratory has shown how the relational data management system (System R) can be extended to accommodate the arbitrary length of data items that are commonly encountered in engineering applications. Felippa (1982) has demonstrated how a word-addressable file organization can be simulated with FORTRAN 77 direct access files. The author points out that word-addressable structure is a natural and desirable working-file organization for scientific computing software.

The application of data management for numerical computations is fairly new. Daini (1982) has developed a model for numerical database that arises in many scientific application to keep track of large sparse and dense matrices. The paper presents a generalized facility for providing physical data independence by relieving users from the need for knowledge of physical data organization on the secondary storage devices. Because of the limitation on core storage and to reduces the input-output operations involved in secondary storage techniques, many investigations have been carried out on efficient use of the memory. A detailed survey by Pooch and Nieder (1973) gives various indexing

techniques that can be used in dealing with sparse matrices. Darby-Dowman and Mitra (1983) describe a matrix storage scheme in linear programming. Rajan and Bhatti (1983) present a memory management scheme for finite element based software. Much work is still needed in application of database management in design optimization.

6.2 DATABASE MANAGEMENT SYSTEMS

Several database management systems have been developed or are in the process of being developed. In this section we will review the following systems relative to their applicability to multi-disciplinary design optimization environment:

DELIGHT - Design Language with Interactive graphics and a Happier Tommorow

DATHAN - A data handling program for finite element Analysis

EDIPAS - An Engineering Data Management System For CAD

FILES - Automated Engineering Data Management System

GIFTS - GIFTS Data Management System

GLIDE - GLIDE Language with Interactive graphics

ICES - Integrated Civil Engineering System

PHIDAS - A Database Management System for CAD

RIM - Relational Information Management System

SDMS - A Scientific Data Management System

SPAR - SPAR database management system

TORNADO - A DBMS for CAD/CAM system

XIO - A Fortran Direct Acess Data Management System

Broadly speaking, there are two types of database management systems for engineering computations; context-free and application-oriented managers. The context-free data managers are designed to work as stand-alone systems. In such systems, the applications program executes under the control of DBMS, i.e., the DBMS acts as the main program. The application-oriented managers, on the other hand, are in the form of a library of subroutines that perform all the data handling operations.

6.2.1 DELIGHT

DELIGHT (Nye, 1981) stands for Design Language with Interactive Graphics and a Happier Tommorow. In its philosophy, the DELIGHT system is very close to the GLIDE system (Eastman and Henrion, 1980). DELIGHT is an interacive programming language. It has good extension and debugging capability. It provides high-level graphic commands, a built-in editor and a well-defined interface routines. A single statement, procedure or part of an algorithm can be tested without having to write and load/link a program. The system relies on virtual memory management of the operating system. It is difficult to use the system with large scale programs. Multiple users are not allowed in the system.

6.2.2 DATHAN

DATHAN (SreekantaMurthy and Arora, 1983) stands for data handling program. It was written mainly for finite element analysis applications. The program has some basic in core buffer management scheme. The program has capability to store permanent and temporary data sets. Substructure files can be arranged quite easily with same

data set names for different substructures. Both integer and real data types can be handled. Drawback of the system is that the user has to keep track of the location from which a new data set has to begin. The system has data manipulation commands which are simple to use. The commands can be given using FORTRAN call statements.

6.2.3 EDIPAS

EDIPAS (Engineering Data Interactive and Analysis System); Heerema, and van Hedel, 1983 is a tool for data management, analysis, and presentation. The data management part provides a utility to initalize a project database, input programs to load data from files into database under user controls, and a set of routines to extract data from and load data into database in a controlled way. EDIPAS allows users to name a database, a data structure, and data entities. EDIPAS allows user to employ one or more hierarchical levels. The data is stored in entities called blocks. A data block allows matrices, single values and characteristic values as data elements. A database administration support provides initialization of database, access to users, deletion of data structures, audit database contents, and back-up facility. Drawback of the system is that a clear data definition language is not provided. Variable length records are difficult to process. The system does not have a restart facility. The values of data elements have to be either floating point number or string. A data block has to be unique in a data structure.

6.2.4 FILES

FILES (Lopez, 1974) is an automated engineering data management system. It is extremely flexible with respect to the definition of a database and methods of accessing it. Information storage and retrieval may be performed using problem-oriented languages. Hierarchical data structure is provided. For example matrix type of data encountered in finite element application can be organized using hierarchical data structure. The first two levels in hierarchy may contain pointers to the third level data containing actual matrix data. The program allows dynamic memory allocation. Data transfer takes place between FORTRAN common block and database. FILES has a data definition language scheme. Drawback of the system is that it does not have data mapping language to specify mapping of data items and arrays to an external device. Data is represented only in the form of tables. Data elements are not allowed in the data definition language. The system requires a distinct data management compiler.

6.2.5 GLIDE

GLIDE (Eastman and Henrion, 1980) is a context-free database management system. It is designed to provide a high level facility for developing individualized CAD system. It can be viewed as a language, a database management system, and a geometric modelling system. It allows users to define new record types known as FORMS that consist of a set of attribute field. It provides primitive data type set to organize a database. It provides excellent geometric modelling system or a graphic

system. Drawback of GLIDE is that it does not allow multi-dimensional arrays. It does not support multiple users simultaneously.

6.2.6 GIFTS

GIFTS (Kamel, McCabe and Spector, 1979) is an interactive program for finite element analysis. It is a collection of modules in a program library. Individual modules run independently and communicate via the unified database. The database manager processes requests for opening a file, closing a file, storing data set in a file, and retrieving data set from a file. The program has memory management scheme. Each data set is stored in a separate random access file. Paging is carried out within the working storage. A unique set of four routines is associated with a data set for opening and initializing the working storage, for reading a data set, for creating/modifying the data set, and for realizing the working storage. Drawbacks of the system is that for every new data set to be created four new routines have to be written. Each data set is associated with a separate common block, thereby increasing the number of common blocks in the system. The data manager is application dependent and cannot be used as a stand alone system.

6.2.7 ICES

ICES (Integrated Civil Engineering System; Roos, 1966) is a computer system designed for solving Civil Engineering problems. ICES consists of a series of subsystems each corresponding to an engineering discipline. It provides a Problem Oriented Language which can be used to write subsystem programs (e.g., Coordinate geometry program, Stress analysis program). Command Definition Language is used by a programmer

to specify the structure and required processing for each subsystem commands. A Data Definition Language is used to specify the subsytem data structure. It uses its own programming language called ICETRAN (ICES FORTRAN) and has a precompiler which translates ICESTRAN to FORTRAN statements.

Dynamic data structuring capability is provided in the system which helps to organize dynamic arrays in the primary memory. Hierarchical data structure is used for data modelling. Three hierarchical levels: equivalence class, members, and attributes are provided. Data is stored on secondary storage using random access files. Data management program uses buffers to convert logical records to physical records. Identifier is supplied by the programmer which is a pointer giving the position on secondary storage of physical record. The programmer has a choice to store data using dynamic arrays or using data management system depending on amount and use of the data. Drawback of the system is that it uses precompiler ICETRAN to convert to FORTRAN program instead of directly to machine language. Physical storage of data requires knowledge of address and pointers which the programmers have to give. Data independence is thus not achieved. Only three levels of hierarchy is adopted and it is difficult to extend to many levels of hierarchy.

6.2.8 PHIDAS

PHIDAS (Fischer, 1979) is a data management system specially designed for handling a collection of structured data on minicomputers. The architecture of PHIDAS is in accordance with the ANSI-3 schema. It has an

external subschema based on network model of CODASYL and an internal schema for physical tuning particularly suited for engineering database. The data description language is provided to describe schema and subschema. PHIDAS also has a storage structure description language. Data manipulation language is FORTRAN call statements to subroutines. Drawback of the system is that it is difficult to represent matrix type data.

6.2.9 RIM

RIM (Comfort, Erickson 1978) stands for Relational Information Management system. RIM has capability to create and modify data element definition and relationships without recompiling the schemes or reloading the database. RIM provides capability to define new types of data for use in special application such as graphics. RIM supports three types of data: real, integer, and text. Data definition and data manipulation languages are available to define or manipulate relations. The user has capability to project, intersect, join and subtract relations. RIM has good query language. RIM's modification commands permit the user to update relation definition, change data values, attribute names, delete tuples and delete the entire relation. Utility commands such as LOAD, and EXIT are provided to load a new database and close an existing database. Drawbak of RIM is that it does not allow relation having row size more than 1024 computer words. It does not provide an easy to use matrix manipulation routines. The application oriented FORTRAN call statements do not have capability to define attributes, relations, rules, etc., required in defining a schema. The system does not support management of a temporary database.

6.2.10 SDMS

SDMS (Massena, 1978) is a database management system developed specifically to support scientific programming applications. It consists of a data definition program to define the form of databases, and FORTRAN compatible subroutines to create and access data within them. Database contains one or more data sets. A data set has form of a relation. Each column of a data set is defined to be either a key or data element. Key must be a scalar. Data elements may be vectors or matrices. The element in each row of the relation forms an element set. Temporary database capability that vanishes at the end of a job is provided. A scientific data definition language provides a program-independent data structure. Both random and sequential access of data set is possible. Data elements include scalars, fixed and variable length vectors, fixed and variable-size matrices. Data element types include text, real and integer. Drawback of the system is that it does not have a query language. Generalized database load/unload is not available. Double precision data type is not allowed. The system is implemented only on Cyber series computers.

6.2.11 SPAR

SPAR (Whetstone, 1977) computer program is a collection of processors that perform particular steps in finite element analysis procedure. The data generated by each processor is stored on a database compiler that resides on an auxillary storage device. Each processor has a working storage area that contains the input and the computed data from the processor. Allocation of spaces in the storage area is a

problem dependent and is dynamically allocated during execution. Data transfer takes place directly between a specified location on disk using a set of data handling utilities. SPAR database complex is composed of 26 data libraries or data files. Libraries 1 to 20 are available for general use. Libraries 21 to 26 are reserved for temporary and internal use. The database manager uses a master directory to locate the table of contents which in turn is used to locate the data sets in the database. Physically, the auxillary storage is divided into sectors of fixed size and each read/write operation begins at the beginning of a sector. Drawback of the system is that it does not provide either hierarchical or relational data structure. Excessive fragmentation may take place if the sector size does not happen to be an integral multiple of the data that is stored.

6.2.12 TORNADO

TORNADO (Ulfsby, Steiner and Oian, 1979) is a DBMS system developed for CAD/CAM application. It is a CODASYL network system written in FORTRAN with a very useful for handling complex data structures. It handles variable object length and dynamic length records. System allows different data types - integer, real, character, double precision, double integer, complex and logical data. The system has easy to use data definition language and data manipulation language. TORNADO system is highly portable. Data in the database can be accessed by name. There is no restriction on data set types and allows many-to-many relationships. Drawback of the system is that the size of a data object defined by the system is limited by the largest integer value

that can be represented in the computer. The size of the database is limited by the maximum size of a file. A multi-file version is not available. Matrix data is difficult to handle in the system.

6.2.13 XIO

XIO (Ronald, 1978) is a set of subroutines that provides generalized data management capability for FORTRAN programs using a direct acces file. The system allows arrays of integer, real double precision and character data storage. Both random access and sequential access of data is provided. Variable length record I/O is allowed in the system. Bit map scheme is used to identify the unused space for storage of data to minimize disk storage requirement. The program allows restart facility using saved file following completion of a partial execution or after a program termination. Drawback of the system is that it can only operate on IBM360 or DEC PDP11 computing system. The system does not provide data definition language. It does not provide either hierarchical or relational data structures.

VII. DISCUSSION

Database management concepts in design optimization are discussed in the report. The need for a database management system is emphasized. The special features in design optimization such as iterative solution procedure, uncertainty of solution, suitability for interactive computation, incorporation of finite element analysis are described. Data used in design optimization is discussed. It is shown that large amount of data is used in design optimization and data organization is quite essential to ease designer's task of data management. Also, it is shown that amount of computation in design optimization is large and efficient data management is essential. Various terminologies used in database management field are listed in an appendix. This is done primarily because different groups in database management area use different terminology. Only well-accepted terminology is given.

Database organization for design optimization is discussed. The importance of classification of database on the basis of projects, subsystems etc., is discussed. Data organization suitable for modular program approach is presented. Different data models - hierarchical, network and relational are discussed. Examples from both design optimization and finite element analysis are chosen for demonstration purpose. Data model suitable for design optimization is presented.

Various concepts of file structure are presented. The method used for storing, retrieving and modifying data is described in detail. Various techniques such as blocking, record length, paging, pointers,

addressing, etc., are described. A file structure suitable for design optimization is also discussed.

A review of literature on database management in engineering application is made. Several database management systems have been developed. Features of each system are reviewed. Favorable features and drawbacks of each system are noted. It is seen that the field of database management in computer-aided design is quite recent. There is substantial scope for enhancements in the area.

APPENIDIX A

TERMINOLOGY USED IN DATABASE MANAGEMENT

The terminology used in database management for describing various ideas differ considerably from one group to another and even from one time to another within the same organization. It is therefore necessary to list the various terminologies used in the report which are taken from most widely accepted sources. They are grouped into three categories - Hardware terminology, Logical data terminology and Physical storage terminology. They are listed in the following paragraphs in an alphabetical order.

A.1 HARDWARE TERMINOLOGY

Auxiliary Storage: Storage facilities of large capacity and lower cost but slower access than main memory. They are also referred to as Peripheral or Secondary storage devices. Usually accessed via data channels, in which case data is stored and retrieved by physical record blocks. They include magnetic tape and disk units, drums and other devices used to store data.

Cell: is used as a generic word to mean either track, cylinder, module or other zone delimited by a natural hardware boundary such that the time required to access data increases by a step function when data extends beyond a cell boundary (Martin ,1977).

Cylinder: An access mechanism may have many reading head. Each head can read one track. A cylinder refers to a group of tracks that can be read without moving the acces mechanism (Martin, 1977).

Direct-Access Storage Device: IN direct-access storage device, access to a position for storage or retrieval of data is not dependent on the position at which data was previously stored or retrieved. It

is also called random access device.

Input-Output (I/O) Device: An auxiliary storage device connected to the CPU by a data channel.

Main Memory: A fast, direct-access, electronic memory hardwired to the central processing unit. It holds machine instructions and data that can be accessed in a time of the order of nanoseconds. It is also, referred to as 'core' ,'main storage', or 'internal memory'.

Module: A module of the peripheral storage device is a section of hardware which holds one volume, such as one spindle of disks (Martin,1977).

Storage Device - Logical (Logical File, Memory Device, Name Space, or Logical Address Space: A subset of the storage space that is treated as a named entity by the operating system for purpose of allocating and releasing storage resources during the execution of a run unit (task). The term is most often applied to auxiliary storage facilities

Storage Facility: Hardware available to store data at a computer installation.

Track: A track on a direct-access device contains data that can be read in a single reading without the head changing its position (Martin, 1977).

Volume: A volume is normally a single physical unit of any peripheral storage medium such as tapes, disk packs, or cartridges (Martin, 1977).

A.2 LOGICAL STORAGE TERMINOLOGY

Arithmetic Data: An arithmetic data item has a numeric value with characteristics of base, scale mode and precision; e.g., fixed point data (integer), and floating point data (real and double precision).

Attribute: Properties of entities are called attributes. Attributes associate a value from a domain of values for that attribute with each entity in an entity set. For the entity 'finite element', length of the element, number of nodes of the element, element properties, etc., are its attributes.

Creation: It involves adding new files to the database, initializing the files (i.e., file table definition), data validation, deciding file types, etc.

Data Aggregate: A data aggregate is a collection of data items within a record. Data aggregates may be a vector or repeating groups.

(a) Vector is a one dimensional ordered collection of data items.

Example: Node numbers of a structure

(b) Repeating Group is a collection of data that occurs repeatedly within a data aggregate.

Example: Degrees of freedom of an element.

Degrees of freedom for the element appear in multiples of node numbers.

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots D_{1i} \quad D_{2i} \quad D_{3i}, \quad i = 1,n$$

where i = node numbers and n = total number of nodes.

Data Definition Language (DDL): It is a set of commands that enable users of DBMS to define data structures to store the data. All data that is to be managed by the DBMS must follow the rules laid down in data definition language. A DBMS must provide DDL to specify conceptual scheme and some of the details regarding the implementation of the conceptual scheme by a physical scheme. It is not a procedural language, but a notation for describing relationships among types of entities in terms of a particular data model.

Data Independence: It refers to the independence between physical and logical data structures. Physical data structure can change without affecting the user's view of the data when we have data independence. Similarly, logical data structure can change without affecting the physical data structure.

Data Item: A data item is the smallest unit of named data. It is also referred to as the data element or field. Each data item has a unique representation. The data item can be any of the following types: arithmetic (integer, real or double precision real), or character string (character, bits).

Data Library: A named collection of data sets residing on a permanent storage device. It is the most complex data structure upon which a global database management system operates (Felippa, 1980).

Data Manipulation Language(DML): It is a set of permissible commands that are issued by users or application programmers to the DBMS to carry out storage, retrieval or manipulation of data. The DML represents interface between the application program and the

database management system. Thus the data managed by the DBMS can be accessed and processed through the use of DML. It can be an extension of the host language.

Data Model: Data model is a representation of the conceptual scheme for the database. Generally a data definition language which is a higher-level language is used to describe the data model. Examples of data model are hierarchial, network and relational.

Data Set: An ordered collection of logically related data items arranged in a prescribed manner. Each data set has some control information that can be accessed by a programming system (Martin, 1977).

Data Structure: Logical arrangement of data as viewed by the users or applications programmers.

Database: A database is a collection of the occurences of multiple record types, containing the relationships between records, data aggregate and data elements. It is a collection of data files stored on a storage device.

Database Administrator (DBA) is the brain of the system. It provides interfaces between the various parts of the system, does error recovery, and enforces security measures.

Database Management System (DBMS) The software that allows one or many persons to use and/or modify the database is called a DBMS. DBMS also deals with security, integrity, synchronization and protection of the database.

Database System: The set of all databases maintained on a computer installation (or computer network), which are administered by a common database manager.

Entity: It is a thing that exits and is distinguishable, e.g., finite element.

Entity Identifier: It is necessary for the programmer to be able to record information about a given entity. Also it is necessary for the computer to be able to identify it and have means of finding it in storage unit. Entity identifier must be unique. Example - Element number.

Entity Set: Collection (group) of all similiar entities is referred to as an entity set. Example- all finite elements.

Garbage Collection: The process of locating all pages that are no longer in use and adding them to the list of available space.

Group: A data set containing a special 'owner' or 'master' record (the group directory) and a set of member records (Felippa, 1980).

Instances: The current contents of a database is called an instance of the database.

Interrogation: This deals with identification, selection and extraction of data from the database for further processing. It can be divided into two phases:

a) The process of selection and identification of needed data and extracting it.

b) The processing part which involves computation, display or any other manipulation required including updating parts of the database.

Logical Data Structure: Data in a particular problem consist of a set of elementary items of data. An item usually consists of single element such as integers, bits, characters and reals, or a set of

such items. The possible ways in which the data items are structured defines different logical data structures. Therefore, it is the data structure as seen by the user of the DBMS without any regard to details of actual storage schemes.

Memory Management System (MMS): A system that allocates the available memory to the different entity sets in a program and makes it appear as if more memory is available than what the computer has.

Module: A program that performs an identifiable task.

Programming Language: The language that an application programmer may use; e.g., FORTRAN.

Program Library: A collection of subroutines that perform primitive functions.

Primary Key: The entity identifier is referred to as the key of the record group or strictly it is primary key. Example - Element number.

Query Language: Query is the process of question and answer that can be accomplished using the query language, i.e., Query the database. The commands are generally quite simple and can be used by nonprogramming as well as programming users. These can be interactive commands as well as utilities that can be called from an application program.

Record: A record is a named collection of data elements or data aggregate. When an application program reads data from a database, it may read one complete record at a time.

Schemes: When a database is to be designed, we develop plans for it. Plans consist of an enumeration of the types of entities that the

database deals with, the relationships among the types of entites, and the ways in which the entities and relationships at one level of abstraction are expressed at the next lower level. The term scheme (schema) is used to refer to plans, so we talk about conceptual schemes and physical schemes. The plan for 'view' is referred to as a subscheme (subschema).

Secondary Key: The computer may also use a key which does not identify a unique record but identifies all those which have certain properties. This is referred to as secondary key.

Storage Address (Address): A label name or number that identifies the place where data is stored in a storage device. The part of a machine instruction that specifies the allocation of an operand or the destination of a result.

String Data: String data are either of the type character or bit. The length of the string data item is equivalent to the number of characters (for a character string) or the number of binary digits (for a bit string) in the item.

Subscheme (View): A map of a programmer's view of the data he uses. It is derived from the global logical view of the data - the schema, and external schema (Martin, 1977). It is an abstract view of a portion of the conceptual database or conceptual scheme. A scheme may have several subschemes. These are defined using the data definition language (DDL).

Systems Programmer: A person responsible for installation and maintenance of computer programs.

Vectors: It is a one dimensional ordered collection of data items, all of which have identical characteristics. The dimension of a vector is the number of data items contained in it.

Word: The standard main storage allocation unit for numeric data. A word consists of a predetermined number of byte characters or bytes, which is addressed and transferred by the computer circuitry as an entity.

A.3 PHYSICAL DATA STORAGE TERMINOLOGY

Address: It is a means of assigning data storage locations and subsequently retrieving them on the basis of key for the data.

Bit: An abbreviation of binary digit. The term is extended to the actual representation of binary digit in a storage medium through an encoded two-state device (Felippa, 1980).

Byte: A generic term to indicate a measurable portion of consecutive binary digits. The smallest main storage unit addressable by hardware. In machines with character addressing, byte and character are synonymous (Felippa, 1980).

Character: Member of a set of elementary symbols that constitute an alphabet interpretable by computer software. A group of consecutive bits that is used to encode one of the above symbols.

File: A file is a named collection of all occurrences of a given type of logical records. It is also a collection of data sets.

Page: A basic unit of primary storage; also basic transaction unit between primary and secondary storage.

Paging: In virtual storage systems, the computer memory is made to appear larger than it is by transferring blocks (pages) of data or programs into memory from external storage when they are needed. This is called paging.

Pointer: The address of a record (or other data grouping) contained in another record so that a program may access the former record when it has retrieved the latter record. The address can be absolute, relative or symbolic.

Physical Data Structure It is important to distinguish explicitly between logical data structures and the ways in which these structures are represented in the memory of a particular computer. This may be dictated by specific hardware and software systems. The way in which a particular logical data structure is represented in the memory or secondary storage of a computer system is known as storage or physical data structure.

Sequential Access: A serial access storage device can be characterized as one that relies strictly on physical sequential positioning and accessing of information.

Storage: The process of assigning specific areas of storage to specific type of data.

Virtual Memory: The simulation of large capacity main storage by a multi-level relocation and paging mechanism implemented in the hardware.

REFERENCES

Afimiwala, K.A. and Mayne, R.W., 1979, "Interactive Computer Methods for Design Optimization," Computer-Aided design, Vol. 11, No. 4, pp. 201-208.

Bell, Jean, 1982, "Data Modelling of Scientific Simulation Programs," Int. Conf. On Management of data, June 2-4, ACM-SIGMOD, pp. 79-86.

Blackburn, C.L., Storaasli, O.O. and Fulton, R.E., 1982, "The Role and Application of Database Management in Integrate Computer Design," Journal of American Institute of Aeronautics and Astronautics, pp. 603-613.

Browne, J.C., 1976, "Data Definition, Structures, and Management in Scientific Computing," Proc. Of ICASE Conference on Scientific Computing, April 1976, pp. 25-56.

Bryant, J. C., 1978, "A Data Management System for Weight Control and Design-to-Cost", NASA Conference Publication 2055.

Buchmann, A.P. and Dale, A.G., 1979, "Evaluation Criteria for Logical Database Design Methodologies," Computer-Aided Design, pp. 121-126.

Comfort, D. L. and Erickson, W. J., 1978, "RIM-A Prototype For A Relational Information Management System," NASA Conference Publications 2055.

Daini, O. A., 1982, "Numerical Database Management System: A Model," Int. Confer. On Management Of Data ACM-SIGMOD.

Darby-Dowman, K. and Mitra, G., 1983, "Matrix Storage Schemes In Linear Programming," SIGMAP bulletin ACM, No.32, April 1983, pp. 24-38.

Date, C.J., An Introduction To Database Systems, Addison-Wesley, Reading, Mass., 1977.

Derwa, G.T., 1978, "Advanced Program Weight Control System," NASA Conference Publication 2055.

Eastman, C.M. and Henrion, M., 1980, "The Glide Language for CAD," J. Of the Technical Councils Of ASCE, Vol. 106, No. TC1, Aug. 1980, pp. 171-184.

Eastman, C.M. and Fenves, S. J., 1978, "Design Representation and Consistency Maintenance Needs in Engineering Databases," NASA Conference Publication 2055.

Elliott, L., Kunii, H. S., and Browne J.C., 1978, "A Data Management System For Engineering and Scientific Computing," NASA Conference Publications 2055.

Emkin, L. Z., 1978, "ICES Cocepts-A Modern System Approach," Computing In Civil Engineering pp. 89-107.

Felippa, C. A., 1979, "Database Management In Scientific Computing-I General Description," Computers and Structures, Vol. 10, pp. 53-61.

Felippa, C. A., 1980, "Database Management In Scientific Computing-II, Data structures and Program architecture," Computers and Structures, Vol. 12, pp. 131-145.

Felippa, C. A., 1982, "Fortran-77 Simulation of Word-addressable Files," Advanced Engineering software, Vol. 4, No. 4.

Fischer, W.E., 1979 "PHIDAS -a Database Management System for CAD/CAM Software," Computer-Aided Design, Vol 11, No. 3, pp. 146-150.

Fishwick, P.A. and Blackburn, C.L., 1982, "The Integration Engineering Programs using a Relational Database Scheme," Computers In Engg, Int. Comp. Engg. Confer., pp. 173-181.

Fulton, R. E. and Voigt, S. J., 1976, "Computer-Aided Design and Computer Science Technology," Third ICASE conf. on Scientific Computing, April 1976, pp. 57-82.

Galletti, C.U. and Giannotti, E.I., 1979, "Interactive Computer System Functional Design Of Mechanisms," Computer-Aided Design.

Grabowski, H., Eigner, M. and Rausch, W., 1978, "CAD Data-Structures For Minicomputers," Third Int. Conf On Computers and Engg., CAD 1978, pp. 530-548.

Grabowski, H. and Eigner, M., 1979, "Semantic Datamodel Requirements and Realization with a Relational Data Structure," Computer-Aided Design, Vol. 11, No. 3, pp. 158-167.

Haskin, R. L. and Lorie, R. A., 1982, "On extending the Functions Of a Relational Database System," Int. Conf. On Management of Data ACM, 1982, June 2-4, pp. 207-212.

Haug, E.J. and Arora, J.S., 1979, "Applied Optimal Design," John Wiley and Co., 1979.

Heerema, F.J. and van Hedel, H., 1983, "An Engineering Data managemet System for Computer-Aided Design", Advanced Engineering Software, Vol. 5, No. 2, pp. 67-75.

Jefferson, David K. and Thomson, Bernard M., 1978, "Engineering Data Management: Experience and Projections," NASA Conference publication 2055.

Jenne, R. L. and Joseph, D. H., 1978, "Management of Atmospheric Data", NASA Conference Publication 2055.

Jumarie, G., 1982, "A Decentralized Database via Micro-computers a Preliminary Study," Computers in Engineering, Int. comp. Engg. Confer. ASME, pp. 183-187.

Kamel, H. A., McCabe, M. W. and Spector, W. W., 1979, GIFTS5 System Manual", University of Arizona, Tucson.

Kunni, T. I. and Kunn, H. S., 1979, "Architecture of a Virtual Graphic Database System For Interacive CAD", Computer-Aided Design, Vol. 11, No. 3, 1979.

Lafue, G., 1978, "Design Database and Data Base Design," Third Int. Conf. On compters in Engg. and Building Design CAD78, Brighton Metropole, Sussex, U.K., 14-16 March 1978.

Lafue, G.M.E., 1979, "Integrating Language Database for CAD Applications," Computer-Aided Design, Vol. 11, No. 3, 1979, pp. 127-129.

Lopatka, R. S. and Johnson, T. G., 1978, "CAD/CAM Data Management Needs, Requirements and Options," NASA Conference Publications 2055.

Lopez, L.A., 1974, "FILES: Automated Engineering Data Management System," Computers in Civil Engineering, Electronic Computation, pp. 47-71.

Lopez, L.A., Dodds, R.H., Rehak, D.R. and Urzua, J.L., 1978, "Application of Data Management to Structures," Computing in Civil Engineering, pp. 477-498.

Martin, J., 1977, "Computer Database Organization", Prentice-Hall. Inc., Englewood Cliff, N.J.

Massena, W. A., 1978, "SDMS - A Scientific Data Management System," NASA Conference Publication 2055.

Nye, W., 1981, "DELIGHT- Design Language with Interactive Graphics and a Happier Tomorrow," Electronics Research Laboratory, University of California, Berkeley, CA 1981.

Pooch, U. W. and Neider, A., 1973, "A Survey Of Indexing Techniques For Sparse Matrices," Computing Surveys, Vol. 5, No. 2, June 1973, pp. 109-133.

Rajan, S.D. and Bhatti, M.A., 1983, "Data Management in FEM-based Optimization software," Computers and Structures, Vol. 16, No. 1-4, pp. 317-325.

RIM User's Guide, 1980, Academic Computer Center, University of Washington W33, Jan 1980.

Ronald, D. P., 1978, "XIO-A Fortran Direct Access Data Management System," NASA Conference Publication 2055.

Roos, D., 1966, "ICES System Design", The M.I.T. Press, Massachusetts.

Roussopoulos, N., 1979, "Tool for Designing Conceptual Schemata of Databases," Computer-Aided Design Vol. 11, No. 2, pp. 119-120.

Schrem, E., 1978, "Functional Software Design and its Graphical Representation," Computers and Structures Vol. 8, pp. 491-502.

Somekh, E. and Kirsch, U., 1979, "Interactive Optimal Design of Truss Structures," Computer-Aided Design pp. 253-258.

SreekantaMurthy, T. and Arora, J.S., 1983, "A Simple Database Management Program (DATHAN)," Technical Report, Division of Material Engg., The University of Iowa, Jan. 1983.

SreekantaMurthy, T. and Arora, J.S., 1983, "Database Management Concepts In Design Optimization," Technical Report, Division of Material Engg., The University of Iowa, June 1983.

SreekantaMurthy, T., Reddy, C.P. and Arora, J.S., 1983, "User's Manual For Engineering Database Management System EDMS," Technical Report, Division of Material Engg., The University of Iowa, Oct. 1983.

Ulfsby, S., Steiner, S. and Oian, J., 1979, "TORNADO:A DBMS for CAD/CAM Systems," Computer-Aided Design, pp. 193-197.

Whetstone, W. D., 1977, SPAR Structural Analysis System Reference Manual, System Level II, Vol. I, NASA CR-145098-1.

ABSTRACT

This report describes concepts and requirements of a database management system for engineering design optimization and, in general, scientific computing. Distinction between database management in business and engineering applications is first hightlighted. General concepts for design of a database management system in scientific computing and, in particular, engineering design optimization are presented. Based on these concepts and requirements, a set of detailed specifications suitable for database management system (DBMS) has been developed. Such a DBMS can be used in the development, implementation and evaluation of database management concepts and methods for design optimization. Some important specifications for the system are: (1) data independence, (2) multiple logical views of the data, (3) memory management, (4) matrix operation utilities, (5) query language for use in interactive sessions as well as applications programs (useful for defining optimization problems), and (6) management of permanent, temporary, global and local databases. These capabilities must be present for design optimization applications. Based on the specifications, a database management system called EDMS (Engineering Database Management System), has been initiated. The system is being developed and integrated into design optimization methods. It will be used in the design, implementation and evaluation of database management concepts for design optimization.

TABLE OF CONTENTS

I. INTRODUCTION

In a previous report (Rajan, Bhatti and Arora, 1983) database management system for structural design optimization was described. The system has been successfully used to solve known structural optimization problems. In another report (Sreekantamurthy and Arora, 1983), concepts of database design - physical and conceptual - have been studied. Existing literature on scientific database management systems has been extensively reviewed. This has been done to determine the suitability of existing systems for use in design optimization environment. In this report, concepts and requirements for a database management system suitable for general engineering design optimization environment are presented.

Considerable work has been done on database management in business type applications. Several books (e.g., Ullman, 1980; Wiederhold, 1977) are avaiable. Whereas some progress has been made in database management in scientific environment (Sreekantamurthy and Arora, 1983), considerable more work needs to be done. A reason for this slow progress in database management for scientific computing is due to a view that database management systems developed for business applications can be adapted for every other type of application. However this view is not correct as can be seen from the comparisons given in Table 1 (Felippa, 1979, 1980).

There is a functional separation between business and scientific computing that dictates the need for database management systems for scientific applications. Most scientific computing problems can be characterized as "large computing-large I/O". In comparison, business

Table 1

SCIENTIFIC Versus BUSINESS DATABASE MANAGEMENT

Scientific Computing	Business Computing
The data managed is static as well as dynamic. There is a need for frequent updating and deletions.	The data managed is mostly static.
Unit of I/O is large and effects large segments of the database	Unit of I/O is small.
Main memory requirements are high, requiring careful allocation of resources.	Memory requirements are small.
Establishment of relations is dynamic and depends on usage and the user. There is no need to save these relations because they change from application to application.	Relations are known and static. They are established by programmers. They hold good for general users and can be managed by the system
Emphasis is on ease of establishment of relations.	Emphasis is on storing relations.
Need for dynamic altering of logical view of data, i.e., same data can be viewed in more than one logical order.	The organization of logical record is determined at the time of data definition and is viewed in the same order.
Deals with numeric data and is word oriented.	The data managed is character and byte oriented.

problems can be characterized as "small computing-small I/O". Also, the nature of the I/O in two applications differs significantly. Initially, all business applications were termed to be "small computing-small I/O" and scientific applications "large computing-small I/O". The former remains valid today but the later has changed to "large computing-large I/O" with the integration of various disciplines aimed at solving a problem. This change rules out the possibility of using database management systems that are developed for business applications.

Most noticeable trends in scientific computation are the increase in size and complexity of programs, and integration of software from differing scientific disciplines. This trend combined with the shortcomings of existing database management systems dictates the need for a scientific database management system that posseses certain distinct features:

a) That it be a "general-purpose" system which can be integrated with any other existing or new application systems.

b) That it possess memory management which would translate user defined logical structures to physical structures, as well as be responsible for optimizing I/O.

c) That it possess built-in utilities that are commonly used, and provision to develop utilities as the need arises.

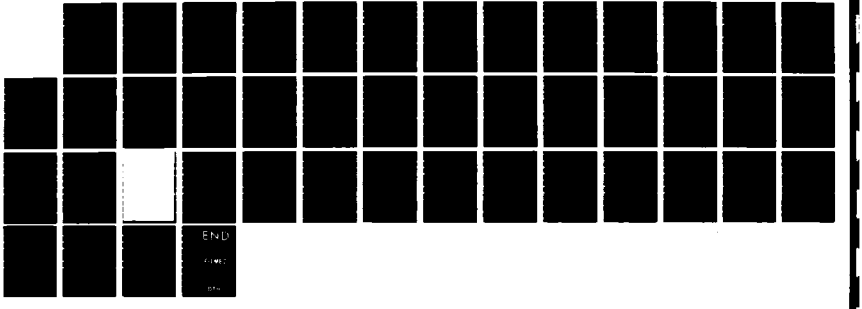
d) That it be developed in a language which is widely used in the area.

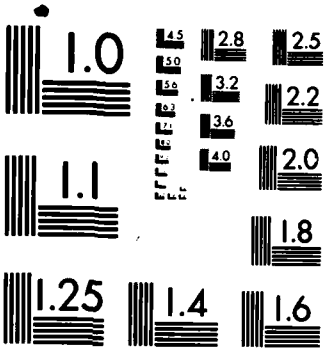
e) And it should be implementable with minor modifications on most computer systems, i.e., portability of the system is essential.

Design of complex systems of the future will require sophisticated software. The process will be automated to some extent. Optimum design technology will play a key role in this process. Proper management of data will be a "must". The database management system will be the core of the design optimization software. The system must be such that it facilitates the development, enhancement and modification of the design optimization software.

Purpose of this report is to present some general concepts and requirements for a desirable database management system in design optimization environment. Based on the specifications given here, development of a database management system has been initiated. The system is being incorporated into design optimization programs to completely develop and evaluate database design and management methods.

AD-A162 212 DATABASE MANAGEMENT IN DESIGN OPTIMIZATION(U) IOWA UNIV 3/3
IOWA CITY APPLIED-OPTIMAL DESIGN LAB
T SREEKANTAMURTHY ET AL 30 OCT 83 CAD-SS-83-17
UNCLASSIFIED AFOSR-TR-85-1083 AFOSR-82-0322 F/G 5/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

II. CONCEPTS AND REQUIREMENTS

2.1 Introduction

It can be seen from the review of the literature that existing database management systems will not satisfy needs of engineering design activity. The context-free systems [GLIDE (Eastman and Henrion, 1980), and DELIGHT (Nye, 1981)] are not suitable as it is impossible to add new capabilities to the system by users from different fields. Only originators of the program can add, modify and extend the system. Therefore, application-oriented managers are more suitable for engineering design act-ivity. In this regard RIM (1980) and GIFTS (Kamel, McCabe and Spector, 1979) data managers come closer to satisfying the requirements. However, as noted earlier (Rajan, Bhatti and Arora, 1983; Sreekantamurthy and Arora, 1983) they also have certain drawbacks. What is needed is a set of standard routines that allow designers to create data structures, manage the memory, and store, retrieve and manipulate the data. The DBMS should also have standard utilities to locate and retrieve data records with values for certain attributes matching the given value. Some of these operations are extensively used in the management of various business-type databases. These can be utilized here also.

In the engineering design environment, there are several unique requirements that must be satisfied by a good database management system. The system should manage not only the permanent database but it must also create and manage the temporary databases in a particular application program. This requirement is absolutely necessary as design of various systems from different fields requires development of special

purpose application programs. Development of such programs and data handling must be facilitated by the DBMS routines. With these general requirements in mind, a set of specifications has been developed for such a DBMS. These are presented later in this chapter. The system is called EDMS which stands for Engineering Database Management System.

2.2 Database Organization

To justify the effort and the expense of implementing the EDMS, it should stand the test of time and place.

TEST OF TIME: The time and expense used in implementation of the EDMS should be returned well before the system is outmoded. Outmode is a certainty but the time lag between EDMS implementation and its outdating can certainly be increased by introducing a high level of flexibility.

TEST OF PLACE: The cost of development of a database is high but it can be justified if it is compatible with different computer systems. This is necessary because in scientific environment several computer systems may be used.

The database, especially in a large organization should be both hierarchical as well distributive. The need for distributive organization stems from the fact that many organizations employ different computer systems and if EDMS is restricted to only one system it results in loss of flexibility. Distributed database is nothing but distribution of data libraries over several systems and accessible through one system. However this, if not needed, can be made

optional. The EDMS residing on one system would be in a position to access parts of the database that are residing on some other system. This is very useful in an environment where there are several micro, mini, and main frame computers. This flexibility however introduces complexity, requiring implementation of communication links between various systems. The problem can be tackled once the basic software has been developed and tested.

The distribution dealt only with data libraries. However, they are not the only things that constitute a DBMS. Figure 1 shows a general layout of the DBMS and a database. The EDMS should be organized in a hierarchical manner. The head of the hierarchy would be the (Database Administrator (DBA)). The interaction between various users and the database is conducted through DBA. DBA is a processor that understands the logical data structures used by the users. It maps them into physical structures, stores the data into the database and retrieves it. The DBA is the most complex part of the EDMS, i.e., it is the brain of the system. Next in the level of hierarchy would be the databases, local and global. These two are at the same level. At the next level is the data library. Several data libraries constitute a global or local database. A data library is comprised of data sets. Data sets are groups of related data items which may be accessed as a unit or sub-unit. These sub-units constitute the lowest level of hierarchy and are units of physical transfer (record/block/page).

The logical view of the database is slightly different and the end users are restricted to this view only (see Fig 2.). The logical view consists of a global and local database after the DBA. The DBA to them

is a black box with multiple entry points. (The entry points are nothing but commands that are supported by the EDMS). The users must be aware of the existence of the global and local databases, so that they can store data into and retrieve data from them. They should have access to the index of the global database and also to their own local database. This is needed to enable the user to verify the existence of various libraries and data sets contained within them. Once they provide information about the data library and the data set they have some flexibility at the next level in accessing or storing data by means of logical structures. They can do that without worrying about the physical organization of the data.

Any user of the database must go through the DBA to get to the database. Some of the functions performed by the DBA are outlined below, disregarding the implementation details.

(i) Map Logical Data Structures into Physical Data Structures and Vice Versa.

Given a logical data structure and the database, data library, data set names, and the data aggregate, the DBA gets record(s) from the database or stores them into it. The importance is in conversion from logical to physical and physical to logical structures.

(ii) Allocation of Secondary Memory Storage.

Allocation of secondary storage deals with initializing data libraries and data sets.

(iii) Protect Database from Unauthorized Users.

The DBA has an access matrix which is a list of users of the

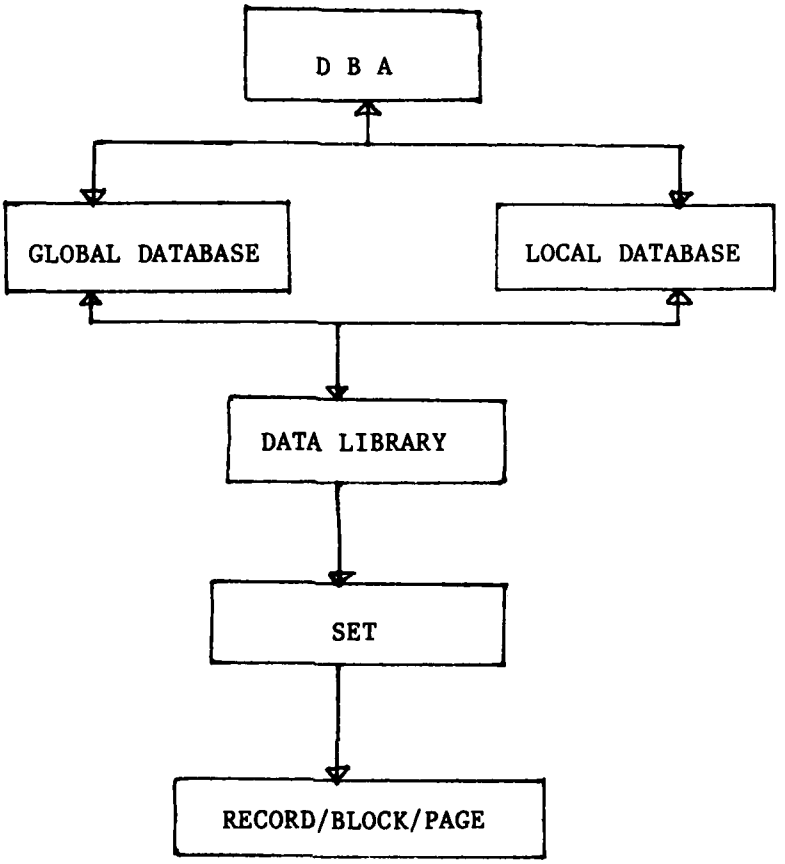


Figure 1. General Layout of a Database

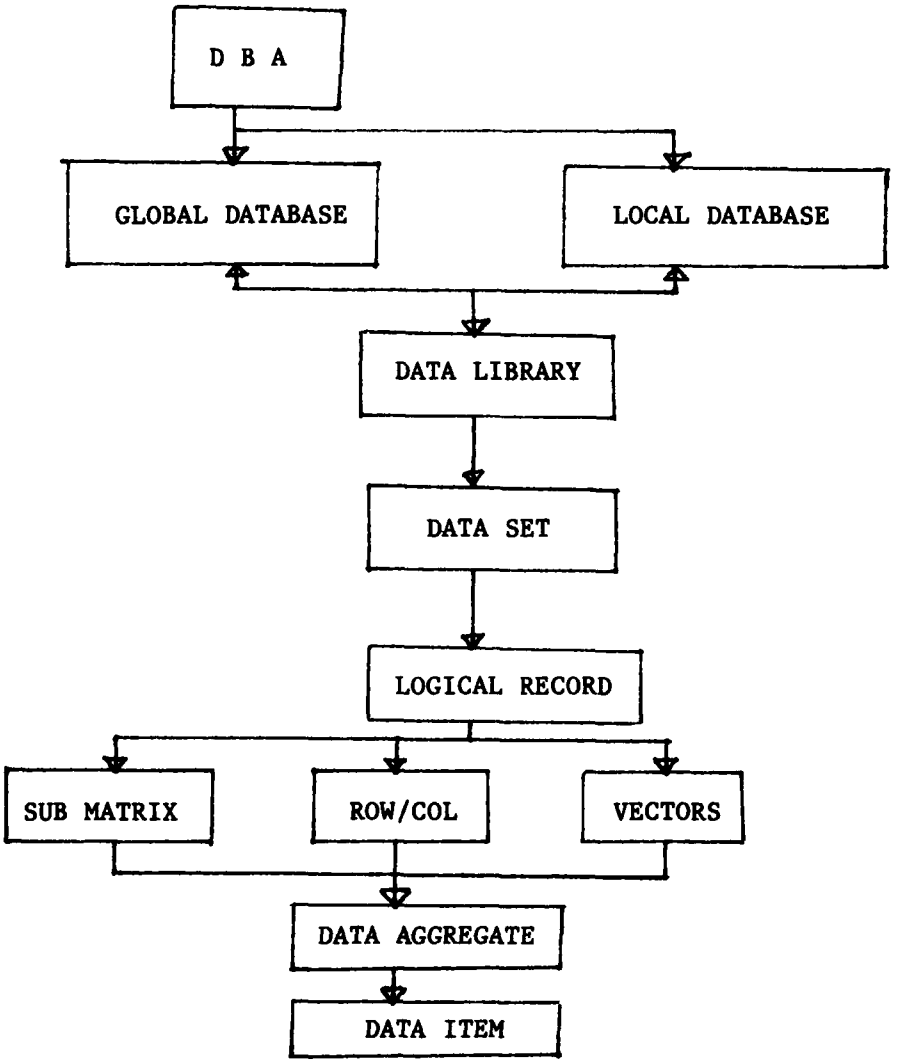


Figure 2. Logical View of a Database

database and all the libraries in the database. This matrix contains information about who can access what libraries. If a user tries to access a library without access rights the DBA protects the database.

(iv) Keep Track of the Indices of Data Libraries as well as Data Sets in each Library.

Given data library and data set names the DBA should map into the physical location of the starting point of the data. To achieve this the DBA has to maintain a list of the data set and their physical locations.

(v) Prepare Data Libraries for I/O

Once the physical location is determined the record that is needed should determined, positioned and transferred to the main memory. The data so transferred may not be in the format of the logical structure required so a conversion might have to be performed.

(vi) Error Detection.

There are several types of errors that can occur. The user should be given sensible reasons for the error and if possible any solution. The most common errors might be data types or wrong data set name in a given library or wrong data library.

(vii) Maintenance of Local Databases.

Maintenance of local databases is very important. It introduces a little distributive feature, because each user may have zero or more local databases. There has to be efficient interaction between global and local databases.

(viii) Support a Query Language Processor.

The user of query language is probably the most demanding user. Such users have to interact with the EDMS in an English like language at the system level. This requires DBA to have a processor that can process the requests and format the data to satisfy the requests. The commands should be processed when the DBA is invoked from an interactive device, or by CALL statements in an application program. The DBA should have syntactic analyser so that commands can be processed. All the above features form the query language processor.

(ix) Page Table and Page Replacement Algorithms.

Since the users are blind to the physical storage structures and are provided the data independence (with respect to logical structures), some memory management has to be performed by the DBA. This involves maintenance of page table, as well as use of page replacement algorithms.

(x) Keep Track of Statistical and Management Information.

The introduction of physical to logical mapping and memory management implies complexity. Since no one method of physical storage or page replacement algorithm can be efficient in a dynamic environment, there should be features (statistical) that monitor the efficiency of given algorithms.

(xi) Maintenance and Updating of Access matrix of all the Users.

The access matrix is primarily used for protection of the database (global). Whenever a new user accesses the database their key is included in the access matrix. Associated with

each data library is a set of keys. The user can access those data libraries that contain the key (given) in their set, i.e., user can access the library.

Let (k1,k2,k3.....kn) be the set of keys associated with the library X. Then a user has the right to access X if and only if

$K_x \in (k1,k2,k3,.....kn)$.

However the set of a library may be null. Such a set is universal and can be accessed by anyone. The set of keys are provided either by the owner of a library or any other authorized manager/staff.

2.3 Classes of Users

For reasons of security and protection of a database the user must be classified, and assigned different priorities and capabilities to access the database. With respect to a scientific database, the users may be classified as follows (see Fig. 3):

(i) Database administrator

(ii) Managers and staff

(iii) Programming users

(iv) Nonprogramming users

Each of these classes of users has a different level of understanding (based on need to know and access requirements). It also requires a different language or interface to the database.

(i) The database administrator requires the maximum amount of flexibility and capability. The capabilities should include

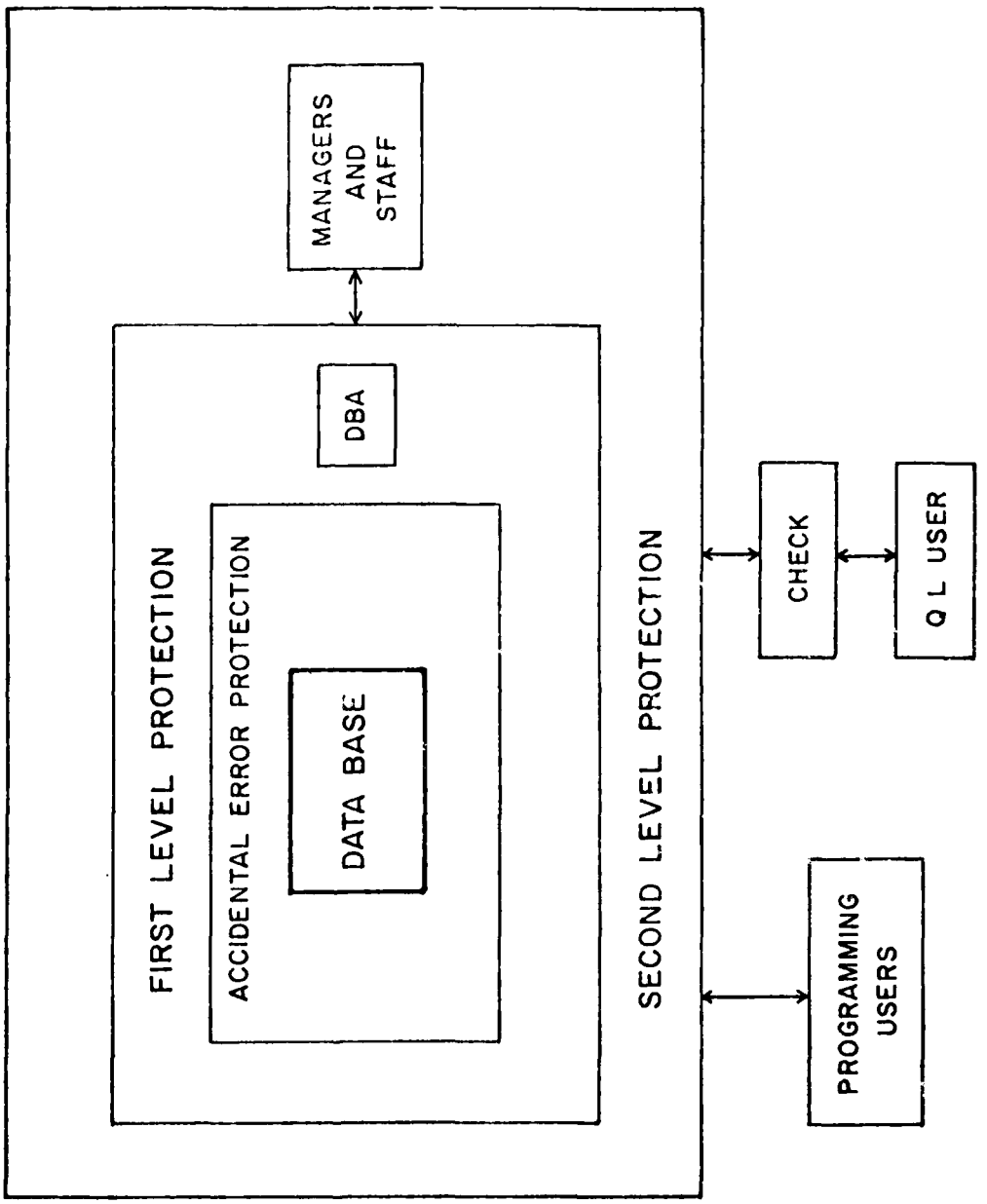


Figure 3. Classes of Users and Access Parts

those of the other three classes of users as well as the capability to control the required security and integrity functions for the database.

(ii) Managers and staff class of users is second in priority level after the database administrator. These users are blacked out from the details of the physical organization of the database, but are aware of the techniques used for storage, protection mechanism, etc. They can insert or delete library key, data sets, and data libraries. They can define the scope of other managers and lower level users to those parts of database for which they have owners rights. They should be aware of the mapping mechanism between logical and physical structures, the statistical information relevant to those parts of the database they are authorized to use. They should have the option of looking at the physical organization of data without database administrator. This is useful to verify the accurate representation of data by DBA or to detect any errors in DBA when storing data. However it is risky if any of the database is altered without the knowledge of the database administrator.

The remaining two classes of users access the database by either using the data manipulation language (DML) or query language (QL).

(iii) The programming user needs the capability to access specific records within the database through the use of a collection of database primitives (functions). These users must be able to fetch, store and modify data values stored within those

records. The primitives (functions) are to be described in terms of the host language; i.e., make these as extentions of the host language itself. A host language is a language that is supported by the system being used: e.g., FORTRAN.

(iv) The nonprogramming user accesses and modifies the database through the use of generalized query language (interactive question and answers). This query language (set of commands) should be supported at the system level. A user without a program and with minimal input can access data from the database for display, modification, or for graphing. A typical use for this might be to access the set of X,Y coordinates, display them on an HP terminal and make a plot using AUTO.PLOT.

In scientific environment, most of the data access is done with the intention of further computation or using it for graphic display. It is therefore important for the EDMS to support most common computation procedures as system defined routines. It should also support basic plotting routines that the user may use for displaying graphs on various devices.

2.4 Programmer's Interface

The programmers class of users are mainly concerned with the development of specific data storage and extraction, and data manipulation programs in an engineering and research environment. Since this class makes up for the majority of the users of the EDMS, a high level of efficiency has to be attained. The user must be aware of the

various ways of using the data manipulation commands supported by the EDMS. He should be well aware of all the data. The set of data manipulation commands and their formats can be viewed as a programming language which is an extention of the host language.

2.5 Data Manipulation Language (DML)

This consists of a set of FORTRAN subroutines that perform a predetermined task. It will also concern itself with some protection aspects.

The basic DML commands can be classified as

System Dependent Commands

- Open database (files)
- Close database (files)
- Check existence of files(s)
- Delete (files)

File Information Commands

- Get all the data set names contained in a file
- Get the file type (sequential/random)
- Check file status (open/close)
- Check for access rights (owner/nonowner)
- Position the file
- Dump file table contents

Create and Add Commands

- Add a data set to the file
- Add record(s) to the data set
- Delete records from a data set

Delete a data set from a file

Search and Access Commands

Search for a data set

Search for a record number in a data set

Position to a record number

Read record numbers X to Y

Search for the last record in a data set

2.6 Detailed Specifications

The idea behind the EDMS is that its scope should not be restricted only to programs that would be written after its installation, but should be extended to those programs which have already been implemented. The later can be achieved by modifying some routines. This transition (if the programs have been properly structured) should be quite straightforward.

One of the most important features in EDMS should be data independence. No matter what logical data structure (of course finite) is employed by a program, the EDMS or specifically the DBA should be capable of providing a mapping between the logical and the physical data structures.

In scientific oriented programs, much of the data generated and manipulated will be dynamic. This introduces a different kind of complexity which is quite different from the one often encountered in business application data management. This requirement must be satisfied in design optimization environment.

The EDMS should provide facilities that would give the users an option to save data both in the global as well as local databases. The concept of local database is relevant in scientific research and design environment due to the dynamic nature of the database. In a developmental stage the user may want to verify the accuracy or authenticity of the data before making it available to other users. The EDMS would not maintain multiples of unique data libraries. As several users could use the same data library simultaneously, there is a need for local database. A particular library could be copied into a local database. There, it could be used for updating and other purposes.

The dynamic nature of the database dictates that it should have some statistical features. These would survey the efficiency of the physical structures employed in relation to the logical data structures used. The DBA would be prompted if and when a change in the physical ogranization of a data library or libraries is required.

Based on the previous discussions and analyses, a set of detailed specifications has been developed for the EDMS:

1) The data generated by the users should be stored in the appropriate database (local or global) depending on the user specification. The data managed should be retrievable from any of the bases. Flexibility refers to enabling data storage and retrieval in more than one form. Such flexibility must be provided.
2) If the EDMS is implemented on say PRIME 850 then implementation on any other system say IBM 3033 should at the most involve modification of those routines that are system related like file manipulation, data representation, etc. The logic of the

management system should be independent of the system. For example, the FORTRAN complier on PRIME differs slightly from that on IBM. The transition should involve modification of those parts of the code that are within the scope of the difference in compliers.

3) The data could be generated by programs or it could exist in the form of files. The management system should be able to manipulate this data and store it according to the user specification; the data could be stored in more than one logical way. In storing matrices the system should determine the sparsity of the matrix and should develop a scheme to store only those elements that are non-zero.

4) The data managed by the system should be accessible to any authorized user in two ways: (a) program request, and (b) query request.

5) The data managed by the system is considered valuable therefore it needs some measure of protection and security. The protection introduces the concept of authorization. Is the user authorized to access the requested data, and what is the scope of authorization (read only, read/write, etc.)? Security involves preventing an authorized unwary user from destroying the database accidentally or intentionally.

6) Flexibility in expansion of the scope of the data management system is important. Since the system encompasses all the fields of engineering, visualizing each and every need initially may not be possible. In addition, needs may change with time. This implies that adding new capabilities should be made easy. Utilities

supported should be easily replaceable by new and improved equivalent utilities.

7) The file (library) should be able to hold more than one data set (a data set is a named collection of logical records). Each data set may have different logical organization but this should not prevent them from coexiting in the same file. This is quite important because a system under analysis may have data sets of different characteristics. Each characteristic data should be stored separately but could coexist with others. This way all the characteristics are accessible as a unit or individually.

8) Since each data set can be organized in one logical order, it would be disadvantageous to access data in only one way especially in a multi-user and multi-device environment. If a user wants to access data in an order that is different from the order in which it was stored, the managment system should convert and present the data in the user required format. If the data accessed is to be used for computational purposes, then it can be unformatted. However, if it is used in display mode it has to be pre-formatted as the user needs, and the specific device accepts.

9) This is an implementation detail for the capabilities supported at steps 7 and 8. The database administrator should understand or view the data in at least 3 levels:

   (a) conceptual: the global logical view of the data to be managed.

   (b) external: presents variety of views of the data to multiple users; e.g., conversion of physical storage view to logical view and vice versa.

(c) internal: this view describes the physical characteristics of data, i.e., integer, real, character, single precision, double precision, etc., and how these are represented in a particular device.

10) The database if needed should be distributed over a variety of computers, since it is logical to assume that engineering analysis and design involves various fields or departments. It might not be feasible for all departments to work on the same computer, so the system should have the capability to store data on different computer systems but accessible from any one computer. However there is an underlying assumption that "a network exists between all the computers used".

11) The conceptual schema involves a data defintion language (DDL). Initially all logical structures may not be supported, but if and when the need arises for a new data definition, the DDL should be extended to include it.

12) The EDMS should provide a facility to create temporary files during a process and delete them at the end of the process. This is an extremely useful requirement in design optimization. A number of design iterations are usually necessary before final design is obtained. Therefore temporary data files must be created and deleted during each iteration.

13) The commands to use the EDMS should be well defined, the definition should include a description of its parameters and also the scope and functions performed by the command. A need for simplicity is emphasized.

14) Multiple concurrent users should be supported and be able to access any part of the database simultaneously.

15) The EDMS should support a graphic utility

16) Error recovery of the database should be automated so as to ensure the integrity of the data managed.

17) The system must have its own memory management. If the available memory specified is large, EDMS should perform a minimum of I/O.

III. DISCUSSION AND CONCLUSIONS

A basic purpose of this study is to define a DBMS for engineering design applications. It should be the purpose of the system to facilitate development of various design and optimization applications programs by taking over the burden of data management.

Based on the discussion and review of literature a very comphrensive database management system to satisfy this requirment has been defined. It is determined that an application-oriented system is the best for engineering analysis and design optimization activity. It is highly flexible and can be used with any application. A comphrensive set of specifications has been developed for the system. Based on these specifications, a preliminary form of data definition and data manipulation languages have been developed. A user's manual for this implementation is given in the Part IV of this report. The user's manual contains the current capability for data definition and data manipulation. All the specifications have not been implemented yet. The preliminary implementation is being used to evaluate file handling procedures, paging algorithms and memory management techniques. Substantial progress has been made. However, a lot more effort is needed to fully develop and test the system. Distinctive features for the EDMS are:

a) Memory management: The unit of I/O varies widely from application to application and even within the same application. This indicates the need for some kind of memory management that would help optimize the time spent on I/O and speed up execution time. This task cannot be left to the discretion of

the user. Even though most mainframes and mini's support some kind of memory management at the operating system level, it is not available in micros. Therefore, the user is left with a choice of running his programs either on a computer with memory management or reduce the size of his problem. Memory management can also be justified where one is already available especially if the data management system supports databased utilities. However this memory management should be developed such that the user has a control over the size of paging memory and also the size of each segment or page.

The initial implementation of memory management in EDMS consists of fixed number of pages and the page size. Number of pages is determined by the user prior to the execution of the program and is constant for each run. The paging memory is in the form of an array inserted in the common block, viz. MCONTNT (npages,ipsiz). MCONTNT is a short-integer variable. Since the system handles data other that short-integer, other data types are equivalenced to refer to the same array, For example, DIMENSION ILMEM (npages,ipsiz/2) EQUIVALENCE (ILMEM,MCONTNT) where ILMEM is long-integer variable. Depending on the data type appropriate array name is used.

Every paging management needs some kind of page replacement algorithm. The algorithm adopted to determine the page to be replaced is the "least recently used" (LRU). Here a counter is maintained for each page. When the time comes for a page to be replaced the page with the highest counter value becomes the

candidate. The counter of a page referred to any particular instant is set to zero and the counter values of the rest of the pages are incremented by one. The "least recently used" algorithm is based on a time scale and a page with the largest time gap would be replaced. In this system the time is calculated based on the counter value, i.e., higher the counter value the longer the time span. Pages are replaced only when there are no free pages available. Therefore this implies that entire paging memory could be utilized to retain information about a data set. A dirty bit is associated with each page which helps in determining if data in the page is to be written or just replaced. Whenever a file is closed all the data sets contained within the file having pages allocated to them are emptied form the paging memory. Currently there is no provision for automatic recovery in case of abnormal termination, but would be provided in later implementaions. Experience while testing has shown that a fixed page even for a single application has its own drawbacks. This is because the size of data sets usually vary dramatically within each application thereby rendering the page to be either too big for some data sets and too small for others. The way to counter this problem is to divide the memory into two regions; one used for fixed pages and the other for variable sizes or segments. The user can determine the cutoff point. Any data set below the cutoff point uses the segmented memory and those above use fixed pages. The reason for not adopting a complete

segmentation is that it would involve writing garbage collection routines. This poses quite a difficulty even at the operating system level let alone in a system like EDMS. Most of the garbage collection algorithms would render this system innocuous. The garbage collection scheme would not be adopted even when segmentation is implemented, but the problem would be solved by what is called refreshing the segmented memory. When contiguous memory is not available in the segmented memory to bring in new data then all the data contained within segmented memory is written out to the disk leaving the memory blank. Even though this might prove inefficient at times, it is better than garbage collection and does not require any paging algorithm.

b) Logical structures supported in EDMS are of the form of vectors, matrices and tables. A data set is essentially either a matrix or a vector, but a matrix can be ordered in any one of the following forms, viz. column order, row order or submatrices. However, the logical view is not restricted to this order. A data set ordered in, say, column order can also be viewed in row or submatrix order. This provides a flexibility in viewing the data differently based on the need. Also, it simplifies the process of establishing relations between logical records of the same data sets and with those of other data sets.

c) Physical organization: The process of establishing a file or data library and data sets that are to be contained within it, take a logical sequence as outlined below. Prior to any data

transactions, it is the responsibility of the user to define new files, or open an old file. Then the user can either define new data sets or make use of previously defined ones. If the user is interested in generating data and storing it for later use, the procedure is as follows:

Step 1. Define file

Step 2. Define data set(s)

Once the file is defined, Step 2 can be undertaken either immediately or in another application but prior to an attempt to write data in to it.

Step 3. Write data

Step 4. Read, write or update.

The process of data set definition involves setting up the size of data set, type of data that would be stored, and identifying it by a name. Once this is established the information is maintained in the database and operation with the data set is cross checked before the operation is completed. The logical organization of the data has little to do with the actual or physical storage of the data and the user is not at all concerned with this aspect. Whenever data is requested by the user in terms of logical records they are mapped into the physical database and supplied to the user.

d) Utilities: Every system of this nature should support utilities as part of the system. It is preferable that the utilities be developed by the same group that is developing the system in order to exploit the usage of internal routines in performing a

given task efficiently. It was with this idea that we started to identify some utilities which would be commonly used. These utilities can be divided into three groups:

1) Math Utilities: Most commonly used matrix manipulation operations will be included. These operate on data sets stored in the database. The user is relieved of the responsibility of going through a complicated task of performing these operations in piece-wise fashion in case of large data sets.

2) Stat Utilities: This group will consist of some of the commonly used statistical operations.

3) Graphic Utilities: This group will support a graphic package and interface to several hardware devices, i.e., the graphic output could be directed to more than one physical device.

e) Distributed Processing: Currently little has been done in implementing this feature but is envisioned to be included in later implementations. This basically involves usaged of the system over a network of computers by sharing resources, i.e., data residing on one computer would be made accessible to a program running on another computer.

f) Query Language: The data residing in database can be accessed through an application program or through a query language. This could be used to view or display the data. It is envisioned that the query language would consist of interactive commands to query the database. It would also contain commands

that can be used in an applications program. This is an important and mandatory feature of the system if it has to be used in design optimization of systems. Query language can be used to define the optimization problem (cost function, constraints, etc).

g) Help Facility: With a system of this proportion there is a need for a help facility which would enable new users to reference the calling sequence, implication of commonly occurring errors and methods to correct them.

h) Gripe Facility: This feature enables feedback from the users and the difficulities encountered in using the system.

It is concluded that a database management system such as EDMS is a must for developement of various application programs for engineering design, especially optimal design.

EDMS and Design Optimization

A major goal of the present research is to develop database management concepts for engineering design optimization. In this regard considerable progress has been made. Existing concepts have been studied and evaluated. Data needed for design optimization have veen identified. Desirable properties of a DBMS suitable for design optimization have identified. Some of the basic functions of DBMS have been implemented and tested. However, substantial more work needs to be done in the future.

EDMS has been incorporated into an unconstrained optimization algorithm and tested. It has also been incorporated into a quadratic

programming algorithm. The quadratic programming algorithm is a basic tool that is needed in many constrained optimization methods. Therefore, this program will become a core for developing, implementing and evaluating database management methods for design optimization. EDMS is also being incorporated into a commerically available finite element program. The purpose is to use general available analysis capability for design optimization. The idea is that once general analysis capability becomes available, database design methods for optimization can be developed and tested.

General matrix utilities are also being developed. These will operate on the data sets in the database to support basic matrix and vector operations. Such utilities are not generally available in existing DBMS. However, they are extremely useful in developing, implementing and evaluating DBM concepts for design optimization.

REFERENCES

Blackburn, C. L., Storaasil, O. O. and Fulton, R. E., 1982, "The Role and Application of Data Base Management in Integrated Computer-Aided Design", proceedings of the AIAA/ASME/ASCE/AHS 23rd Structures, Structural Dynamics and Materials Conference, New Orleans, LA, May 10-12, pp. 603-613.

Eastman, C. M. and Henrion, M., 1980, "The GLIDE Language for CAD", J. of the Technical Councils of ASCE, Vol 106, No. TC1, pp. 171-184.

Felippa, C. A., 1979, "Database Management in Scientific Computing - I. General Description", Computers and Structures, Vol. 10, pp. 53-61.

Felippa, C. A., 1980, "Database Management in Scientific Computing - II. Data Structures and Program Architecture", Computers and Structures", Vol. 12, pp. 131-145.

Fenves, S. J., 1973, "Design Philosophy of Large Interactive Systems", in Numerical and Compputer Methods in Structural Mechanics, Fenves et al (Eds.), Academic Press, pp. 403-414.

Kamel, H. A., McCabe, M. W. and Spector, W. W., 1979, "GIFTS 5 Systems Manual", University of Arizona, Tucson.

Nye, W., 1981, "DELIGHT - Design Language with Interactive Graphics and a Happier Tomorrow", Electronics Research Laboratory, University of California, Berkeley, CA.

Rajan, S. D. and Bhatti, M. A. and Arora, J. S, 1983, "A Database Management System for Structural Optimization," Technical Report No. CAD-SS-83-9, Computer-Aided Engineering Laboratory, College of Engineering, The University of Iowa, Iowa City, IA 52242.

RIM User's Guide, 1980, Academic Computer Center, University of Washington, W33.

Sreekantamurthy, T. and Arora, J. S., 1983, "Database Management Concepts in Design Optimization", Technical Report No. CAD-SS-83- Design Optimization Laboratory, College of Engineering, The University of Iowa, Iowa City, IA 52242.

Ullman, J. D., 1980, Principles of Database Systems, Computer Science Press, Inc., Rockville, Maryland.

Wiederhold, G., 1977, Database Design, McGraw-Hill, New York.

This subroutine is used to put submatrix data set from user buffer into file defined on LUNIT. The data set has to be in SUBMAT order and file unit 'RA' type.

CALL DSRDPM(LUNIT,TYPE,DSNAME,ISUB,JSUB,ORDER,I,J,DTTYPE)

The subroutine is used to redefine a data set. The row and column dimensions of the data set (size) can be changed using this subroutine. The values of ISUB, JSUB, I, J are the new data set dimensions.

CALL DSRNAM(LUNIT,TYPE,DORDER,DSNAME1,DSNAME2,IERR)

TABLE OF CONTENTS

I. INTRODUCTION

This report describes the subroutines and their functions in the EDMS library. EDMS stands for Engineering Database Management System. The EDMS library has subroutines that can be called from any FORTRAN application program. Main purpose of the EDMS library is to take the burden of managing the data away from the application programmer. The data can be stored and retrieved quite easily using the library subroutines. EDMS has its own memory management. Therefore, a call to EDMS routine to store or retrieve data may not necessarily require a secondary storage I/O operation. This depends on whether the data manager has free space available in the buffer or not.

To use EDMS for managing data in an application program, two types of operation must be carried out: file operations and data set operations. Files and data sets are defined in the classical sense: file is a collection of data sets and data set is a collection of data items such as integers, real numbers, double precision real numbers, characters, etc. EDMS contains file operation and data set operation commands. New files must be defined before data sets can be written in them. Old files must be opened before existing data sets can be read or new data sets can be created in them.

Chapter 2 contains definitions of various arguments used in the subroutines. Chapter 3 contains the file operation commands and Chapter 4 contains data set operation commands.

The EDMS program is stored in the PRIME computer system at The University of Iowa. The program can be invoked using the following

command:

```
ok, SDMS      /* In PRIME A system
ok, EDMS      /* In PRIME CAELAB system.
```

The above command requests the user to supply the application program name. The application program must be compiled and its binary file must be available in the user's file directory. The application program can contain call statements to various subroutines of the EDMS library.

command:

```
ok, SDMS      /* In PRIME A system
ok, EDMS      /* In PRIME CAELAB system.
```

The above command requests the user to supply the application program name. The application program must be compiled and its binary file must be available in the user's file directory. The application program can contain call statements to various subroutines of the EDMS library.

II. DEFINITION OF SUBROUTINE ARGUMENTS

In this chapter, definitions of arguments used in various subroutines are given. These arguments are used in file operation subroutines and data set operation subroutines given in Chapter III. All definitions are given in an alphabetical order.

2.1 ACCESSTYPE:

Specifies whether the file has to be used for read/write/update (R/W/U). Operation update implies read and write. CHARACTER *12

2.2 DORDER:

The order in which the data is to be stored or retrieved, such as, Columns ('COL'), rows ('ROW') or sub-matrices ('SUB'), Upper triangular row ('UTR'), Upper triangular column ('UTC'), Lower triangular row ('LTR'), Lower triangular column ('LTC'). Note: Currently only the order specified in the DSDEFN subroutine is available. That is data can be retrieved only in the form it is stored. CHARACTER*12

2.3 DSNAME:

The name of the data set that is to be defined or has already been defined. A maximum of 12 characters are allowed with the first character being a letter. If the data set appears as a character variable in the calling program, the name of the variable is sent. If the data set name itself is being sent then it must appear in quotes. If job name is to be concatenated with the data set name, then data set name should contain only 8 characters. The data set name after concatenation with job name will be 12 characters long. CHARACTER *12

2.4 DTTYPE:

The type of data stored in the data set: Real ('REAL'), Double Precision Real ('DREAL'), Integer Short ('INTS'), Integer Long ('INTL') or Character ('CHAR'). CHARACTER *12

2.5 END:

The ending location in the data set at which the user wishes to stop the reading or the writing process. The meaning of END depends on the method specified for retrieving data: a) Row-wise - then END = the ending row number, b) Column-wise - then END = the ending column number, c) Sub-matrix - then END = the ending sub-matrix number, d) Part of a column or row - then END = the ending element in the segment, and e) Triangular matrix - then END = the ending row number or column number. (NOTE: When a part of a column or row is required from a data set (ROW or COL data sets only) then use IELM parameter (see IELM paragraph) IELM is set to zero when a complete row or a complete column or a submatarix is stored or retrieved. INTEGER*4

2.6 FNAME:

This is the name of a file that has been already defined or is to be defined. It can have a maximum of twelve characters. The starting characters must be a letter. CHARACTER*12

2.7 FSTAT:

This specifies the status of the file to be defined or is already defined. Status can be either permanent 'PERMANENT', 'PERM', or temporary 'TEMPORARY', 'TEMP'. Temporary status for a file implies that it will be deleted before the program is stopped. CHARACTER*12

2.8 I,J:

The actual dimensions of the data set. If the data set is a one-dimensional array then the value of J is to be set to 1. For a triangular matrix data set, the size of matrix should be given. INTEGER*4

2.9 I1,J1:

The dimensions of the user buffer IBUFF. For one-dimensional array the value of J1 should be 1. INTEGER*4

2.10 IBUFF:

The user buffer (array name) from which the data is to be retrieved or stored back. NOTE: The array must be properly dimensioned in the calling program and the correct data type specification should be used. The array can be either one-dimensional or two-dimensional while using ROW or COL data sets. Two-dimensional array should be used in the case of SUB matrix data sets. The array should be one-dimensional in case of UTR, UTC, LTR, and LTC data sets.

2.11 IELM:

This parameter is used when the user wishes to obtain part of a column or row from the data set (ROW or COL data sets only). IELM is the specific column or row from which a part of column or row is stored or retrieved. STRT and END are the starting and ending element numbers in the row or col IELM. If the user does not wish to take a segment from a column or row then zero value should be entered in the IELM location. INTEGER*4

2.12 IERR:

This parameter returns an error code if error occurs during any call to the EDMS library routines. It is zero if everything goes well. User should check the error condition after returning from the EDMS library routine. INTEGER*4

2.13 IFORMAT:

Specifies if the data to be written to the file is in formatted or unformatted form (FORM, UNFORM). Default is unformatted. CHARACTER*12

2.14 ISUB,JSUB:

The dimensions of the submatrix in a data set. The value of ISUB or JSUB is zero in the case of ROW,COL,UTR,UTC,LTR, and LTC data sets. INTEGER*4

2.15 JOB:

The job name to be assigned to all data sets belonging to a project. The data set name will be concatenated with the job name. Job name may be set to blank if no job name is used. CHARACTER *4

2.16 LUNIT:

This contains the logical unit number on which the file is defined and opened. It is returned to the calling program by the subroutine DFDEFN. This number is subsequently used in subroutine DSGET, DSPUT, etc., for further reference to this file. INTEGER*4

2.17 NDSETS:

This specifies the maximum number of data sets that will be included in this file. Default is 20 at present and can be changed easily. INTEGER*4

2.18 ORDER:

The ordering of the data set: Row ('ROW'), column ('COL') or sub-matrix ('SUB'), Triangular matrix -upper triangular row matrix ('UTR'), -lower triangular row matrix ('LTR'), -upper triangular column matrix ('UTC'), -lower triangular column matrix ('LTC'). This is the order in which the data set is to be stored or retrieved. CHARACTER*12

2.19 PTHNAME:

The specifid UFD or SUBUFD under which the file is to be defined or is already defined. Maximum length of 256 characters. Each set of UFD/SUBUFD must be separated by '>' sign and should be arranged according to the descending hierarchy. Ex:
AEGDDD>SUB.DIR>....>....>....>. CHARACTER*256

2.20 STRT:

The starting position for reading data from a data set. The meaning of the STRT parameter depends on the method by which the user intends to retrieve the data. If the data is to be read a) Row-wise - then STRT = the starting row number, b) Column-wise - then STRT = the starting column number, c) Sub-matrix - then STRT = the starting sub-matrix number (NOTE: Sub-matrices are numbered column wise), d) Part of a column or row then STRT = the starting element in the segment, and e) Triangular matrix then STRT = starting row number or starting column

number. (NOTE: When a part of a column or row is required from a data set (ROW or COL data sets only) then use IELM parameter (see IELM paragraph). IELM is set to zero when a complete row or a complete column or a submatrix is stored or retrieved. INTEGER*4

2.21 TYPE:

This parameter specifies the type of the file that is to be opened or is already open. The file can be random access or sequential access type. TYPE can be either 'RANDOM' (or 'RA') or 'SEQUENTIAL' (or 'SA'). CHARACTER*12.

III. FILE OPERATION COMMANDS

This chapter describes various statements for file operations:

CALL DFDEFN(FNAME,PTHNAME,TYPE,FSTAT,NDSETS,IFORMAT,LUNIT,IERR)

The function of this subroutine is to create a file in the user file directory specified by the pathname and to open it. Before any data is stored on a file, file must be defined using this subroutine. Maximum of 20 files can be defined in the current version.

CALL DFOPEN(FNAME,PTHNAME,TYPE,FSTAT,ACCESSTYPE,LUNIT)

The function of this subroutine is to open the existing file FNAME. The file must have been previously defined by a call DFDEFN. An existing file must be opened before data sets can be retrieved or create new data sets in the file FNAME.

CALL DFCLOS(LUNIT,TYPE,IERR)

The function of this subroutine is to close a file opened on logical unit LUNIT. Before closing the file, the data sets in the file automatically updated with latest modification to data, if such modification has been done.

CALL DFDELE(FNAME,PTHNAME,LUNIT,TYPE,IERR)

The subroutine deletes the file FNAME on logical unit LUNIT. File must be closed before deleting it, using DFCLOS subroutine.

CALL DFCOMP(FNAME,PTHNAME,LUNIT,TYPE,FSTAT,IERR)

This subroutine compresses the file FNAME defined on LUNIT. The empty space formed due to deletion of data sets in a file are removed by moving datasets together. The subroutine helps in proper utilization of the disk space. The file unit number after compressing the file is set to new value.

IV. DATA SETS OPERATIONS

This chapter describes various subroutines related to operations on data sets:

CALL DSDEFN(LUNIT,TYPE,DSNAME,ISUB,JSUB,ORDER,I,J,DTTYPE)

The function of this subroutine is to define the data set details on a file unit (LUNIT). Size, order, data type, etc,, for the data set are defined. Data set must be defined before data items can be stored in it. ISUB and JSUB are dummy values when defining ROW or COL data sets. Maximum of 20 data sets per file unit can be defined.

CALL JOBASN(JOB)

The function of this subroutine is to assign job name to the data sets. The job name will be useful in identifying data sets belonging to different jobs or projects. The user can define same data set name for different jobs. EDMS system concatenates the job name with data set name and the result is stored in the DSNAME argument on return from subroutine DSDEFN. The subsequent reference to the data set name is made using the new DSNAME. This subroutine has to be called only once before calling DSDEFN subroutine.

CALL DSGET(LUNIT,TYPE,DSNAME,STRT,IEND,IELM,DORDER,IBUFF,I1,
J1,IERR)

The function of this subroutine is to get elements of a data set from the file unit (LUNIT), into the user buffer area. Data set must be residing in the file LUNIT. It must have been created by using EDMS library routines.

CALL DSPUT(LUNIT,TYPE,DSNAME,STRT,IEND,IELM,DORDER,IBUFF,I1,J1,

IERR)

The function of this subroutine is to put elements of a data set from user's buffer to the file unit (LUNIT) that has been already defined.

CALL DSGETR(LUNIT,DSNAME,STRT,IEND,IBUFF,I1,IERR)

This subroutine is used to get rows of a data set from file LUNIT into user buffer. The data set has to be in ROW order and file unit must be 'RA' type.

CALL DSGETC(LUNIT,DSNAME,STRT,IEND,IBUFF,J1,IERR)

This subroutine is used to get columns of a data set from file LUNIT into user buffer. The data set has to be in COL order and file unit must be 'RA' type.

CALL DSGETM(LUNIT,DSNAME,STRT,IEND,IBUFF,I1,J1,IERR)

This subroutine is used to get submatrix of a data set from file LUNIT into user buffer. The data set has to be in SUBMAT order and file unit must be 'RA' type.

CALL DSPUTR(LUNIT,DSNAME,STRT,IEND,IBUFF,I1,IERR)

This subroutine is used to put row data set from user buffer into file defined on LUNIT. The data set has to be in ROW order and file unit must be 'RA' type.

CALL DSPUTC(LUNIT,DSNAME,STRT,IEND,IBUFF,J1,IERR)

This subroutine is used to put column data set from user buffer into file defined on LUNIT. The data set has to be in COL order and file unit must be 'RA' type.

CALL DSPUTM(LUNIT,DSNAME,STRT,IEND,IBUFF,I1,J1,IERR)

This subroutine is used to put submatrix data set from user buffer into file defined on LUNIT. The data set has to be in SUBMAT order and file unit 'RA' type.

CALL DSRDFN(LUNIT,TYPE,DSNAME,ISUB,JSUB,ORDER,I,J,DTTYPE)

The subroutine is used to redefine a data set. The row and column dimensions of the data set (size) can be changed using this subroutine. The values of ISUB, JSUB, I, J are the new data set dimensions.

CALL DSRNAM(LUNIT,TYPE,DORDER,DSNAME1,DSNAME2,IERR)

The subroutine can be used to change name of the data set DSNAME1 to a new name DSNAME2.

CALL DSCOPY(LUNIT1,TYPE1,DSNAME,LUNIT2,TYPE2,IERR)

The subroutine can be used to copy a data set DSNAME from file defined on LUNIT1 to a file defined on LUNIT2.

END

FILMED

1-86

DTIC